Fachhochschule Graubünden
University of Applied Sciences

Churer Schriften
zur Informationswissenschaft

Herausgegeben von
Wolfgang Semar Bernard Bekavac, Ivo Macek, Armando Schär

Arbeitsbereich Bachelor of Science
in Digital Business Management

Schrift 169

Codemigration mit ChatGPT

Evaluation von ChatGPT als Tool zur teilautomatisierten Code-übersetzung von COBOL Code zu Python Code

Stefan Banzer

Chur 2023

Churer Schriften zur Informationswissenschaft

Herausgegeben von Wolfgang Semar,
Bernard Bekavac, Ivo Macek, Armando Schär

Schrift 169

Codemigration mit ChatGPT

Evaluation von ChatGPT als Tool zur teilautomatisierten Codeübersetzung von COBOL Code zu Python Code

Stefan Banzer

Diese Publikation entstand im Rahmen einer Thesis zum Bachelor of Science in Digital Business Management.

Referentin: Prof. Dr. Alexandra Weissgerber

Korreferent: Urban Kalbermatter

Verlag: Fachhochschule Graubünden

ISSN: 1660-945X

Ort, Datum: Chur, November 2023

Abstract

Code von einer Programmiersprache in eine andere zu übersetzen, ist ein Teilprozess der Codemigration. In dieser Bachelorthesis wurde untersucht, ob und inwiefern diese Aufgabe mit der Anwendung von generativer KI unterstützt werden kann. Basierend auf einer Literaturrecherche wurde ein Modell und eine Methodik abgeleitet, anhand welchen die Tauglichkeit (teil-)automatisierten Codeübersetzungen bewertet werden kann. Die Evaluation wurde auf das Tool ChatGPT und exemplarisch mit dem Programmiersprachenpaar COBOL und Python durchgeführt. Das Ergebnis hat aufgezeigt, dass ChatGPT keine perfekten, durchaus aber brauchbare Codeübersetzungen generieren kann, die einen guten Startpunkt für die Weiterverwendung darstellen. Basierend auf den verwendeten Kriterien hat ChatGPT den Code im Durchschnitt mit 72-prozentigem Erfolg übersetzt. Um Potenziale und Limitationen des Tools und die Relevanz für die Praxis abschliessend bewerten zu können, muss die Forschung jedoch erweitert werden.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

APR	Automated Program Repair
ChatGPT	Chat Generative Pre-Training Transformer
COBOL	Common Business Oriented Language
DL	Deep Learning
GB	Gigabyte
GPT	Generative Pre-Training Transformer
IoT	Internet of Things
KI	Künstliche Intelligenz
LLM	Large Language Model
ML	Machine Learning
NLP	Natural Language Processing
NMT	Neural Machine Translation
RLHF	Reinforcement Learning from Human Feedback

1 Einleitung

Vor ein paar Jahren waren die Wörter Blockchain und Bitcoin in aller Munde. Im Jahr 2023 sind es die Begriffe Künstliche Intelligenz (KI) und ChatGPT. In der Wissenschaft wird untersucht, wofür KI eingesetzt werden kann und wie dieser Einsatz gewinnbringend sein kann. In der Praxis erhofft man sich, das enorme ökonomische Potenzial, das KI zugeschrieben wird, nutzen zu können (Chui et al., 2023). Effizienzsteigerung, Qualität und Genauigkeit sind die entscheidenden Stichworte. Dabei können die Anwendungsgebiete noch so vielseitig sein. Alleine Chui et al. (2023) untersuchten das ökonomische Potenzial dieser relativ neuen Technologien anhand von 63 Anwendungsfällen, darunter Sales, Marketing, Softwareentwicklung aber auch Human Resources, Risikomanagement oder wie von Zong & Krishnamachari (2022) genannte Anwendungen in Biomedizin und Medizin oder im Texten. Viele der verwendeten Technologien beruhen auf der Verarbeitung natürlicher Sprache, genannt „Natural Language Processing“ (NLP). Fortschritte in diesem Bereich haben dazu geführt, dass leistungsfähige Sprachmodelle, die man „Large Language Models“ (LLM) nennt, entwickelt werden (Rahaman et al., 2023). Eines dieser Modelle ist ChatGPT, wobei GPT ausgeschrieben für „Generative Pre-training Transformer“ steht. Diese Bezeichnung beschreibt, dass basierend auf einem Training des Modells (Pre-Training) neuer Inhalt erstellt und generiert wird (Zhou et al., 2023). ChatGPT ist ein Tool, das einem Chatbot gleicht und vom Unternehmen OpenAI im November 2022 veröffentlicht wurde (OpenAI, 2022b). Innerhalb von 2 Monaten verzeichnete es bereits 100 Millionen Nutzer:innen (Hu, 2023, zitiert nach Zhou et al., 2023). Seither wird ChatGPT für verschiedenste Anwendungsfälle genutzt und von OpenAI weiterentwickelt, darunter für diverse Aufgaben im Bereich der Softwareentwicklung.

1.1 Problemstellung

Die Verwendung von veralteter Software und Code ist weit verbreitet und bringt eine Reihe von Problemen mit sich. Im Unternehmensumfeld kommen dabei oft die Begriffe Legacy Applikationen und Legacy Code auf. Gemäss Weisz et al. (2021) sehen sich viele Unternehmen mit dem Problem konfrontiert, derartige, komplexe Legacy Applikationen zu modernisieren. Der dahinter liegende Prozess, Software in eine neue Zielumgebung zu überführen, wird Softwaremigration genannt.

Migrationen dieser Art können gemäss Gimnich & Winter (2005) umfassend, aber auch auf einzelnen Ebenen durchgeführt werden. Es wird zwischen verschiedenen Arten der Migration unterschieden, wobei das Ziel in der Regel dasselbe ist. Bisbal et al. (1997) beschreiben dieses Ziel als Anpassung der Systeme auf die Geschäftsanforderungen

unter Beibehaltung deren Funktionalität. Die Notwendigkeit dieser Anpassungen ist auf diverse mögliche Probleme mit Altsystemen, etwa veraltete Hardware, geringe Flexibilität von Software oder deren Architektur oder fehlendes Know-how von Personal zurückzuführen (Bisbal et al., 1997). Das Ziel einer Migration ist es, diese Probleme zu lösen, während idealerweise die Notwendigkeit einer vollständigen Neuentwicklung der Software vermieden wird. Um den Migrationsprozess zu vereinfachen und vor allem überprüfbar zu machen, ist es gemäss Gimnich & Winter (2005) unbedingt nötig, dass Altsysteme die Systemfunktionalität eindeutig vorgeben. Dadurch kann nach der Migration vereinfacht überprüft werden, ob die Funktionalität des migrierten Systems beibehalten werden konnte.

Ein wichtiger Teilaspekt der Softwaremigration ist die Codemigration, die laut Gimnich & Winter (2005) im Bereich der Entwicklungsumgebungsmigration angesiedelt ist. Genau wie ganze Systeme, kann auch nur der Code veraltet sein. Kontogiannis et al. (2010) verwenden hierfür den Begriff "Legacy Code". Die Migration solchen Codes in neue Umgebungen oder Programmiersprachen gilt als pragmatischer Ansatz für das Re-Engineering des Codes. Kontogiannis et al. (2010) bezeichnen die Übersetzung des Codes von einer alten in eine neue Programmiersprache als Migration auf tiefster Ebene.

Ein Beispiel im Kontext der Codemigration ist die Programmiersprache COBOL. COBOL wurde Ende der 1950er Jahre entwickelt und findet heute noch vor allem im Bereich von Transaktionen in Geschäftsbereichen wie Finanzdienstleistungen, Verwaltung und Versicherungen Anwendung. Auch wenn der Anteil an COBOL-Code im Vergleich zu populäreren Programmiersprachen als marginal erscheint, sind nach wie vor mehrere hundert Milliarden Zeilen COBOL in Verwendung (Rubens, 2016, zitiert nach O'Hara, 2018). Unternehmen sehen sich typischerweise mit Herausforderungen wie dem Mangel an Entwicklern mit COBOL-Kenntnissen oder der Tatsache konfrontiert, dass die Programmiersprache Ende der 1950er Jahre, ohne das heutige technologische Wissen und Möglichkeiten zu berücksichtigen, entwickelt wurde (O'Hara, 2018). Deshalb bietet sich die Untersuchung einer Übersetzung von COBOL-Code in eine neue, populärere Programmiersprache an.

Wie O'Hara (2018) sagt, ist bereits die Suche nach einer geeigneten Zielsprache eine Herausforderung im Rahmen von Codemigrationen. Kriterien für die Auswahl sind beispielsweise, dass gleiche oder ähnliche Funktionalitäten abgebildet werden können und dass die Sprache weit verbreitet und entsprechend Entwicklern bekannt ist. Nach einer von Stack Overflow (2022) durchgeführten Umfrage mit 71'547 Teilnehmenden lauten die Top-Ten Programmiersprachen wie folgt: JavaScript, HTML/CSS, SQL, Python,

TypeScript, Java, Bash/Shell, C#, C++ und PHP. Weil Python demzufolge die meistverwendete multifunktionale Programmiersprache ist, wird die Übersetzung von COBOL zu Python untersucht. Python wird als eine der am meisten verbreiteten Programmiersprachen angegeben, mit denen komplexe Funktionen oder sogar Backend-Anwendungen programmiert und implementiert werden können. Im Vergleich zu Python (48.07 %) verwenden nur 33.27 % der befragten Personen Java. Dagegen gaben nur 0.65 % an, COBOL zu verwenden (Stack Overflow Developer Survey 2022, 2022).

Das Thema Ressourcen und Expertise in Bezug auf Codemigrationen hat eine grosse Bedeutung. Deshalb haben Chisolm & Lisonbee (1999) als Künstliche Intelligenz (KI) noch kein Thema war, den Gedanken zur automatisierten Übersetzung von Code geäussert. Auch Weisz et al. (2021) untermauern diese Überlegungen mit der Thematik rund um Expertise und Ressourcenknappheit. Neben der Forschung an semi-automatisierter Codeübersetzung mit Transcompilern wurde bereits die Funktionalität von KI-Tools in der Programmierung erforscht. Diverse Ansätze zeigten dabei bereits auf, dass sich auch KI-Modelle wie ChatGPT für das Lösen von Problemen in Bezug auf Coding anbieten.

1.2 Forschungsleitende Fragestellung

In dieser Bachelorthesis wird untersucht, ob und inwiefern sich ChatGPT in dessen leistungsfähigster Version GPT-4 (Stand Juli 2023) für den Einsatz im Prozess der Codemigration bzw. der Codeübersetzung eignet. Basierend auf der vorab beschriebenen Problemstellung wird auf die Migration von COBOL Code zu Python Code fokussiert. Die forschungsleitende Fragestellung, welche im Rahmen dieser Arbeit untersucht wird, lautet entsprechend wie folgt: *„Welche Potenziale und Limitationen weist ChatGPT als unterstützendes Tool für die Code-Migration von COBOL zu Python auf?“*

1.3 Aufbau der Arbeit

Diese Bachelorarbeit ist in acht Kapitel gegliedert. Nach der Einleitung wird im zweiten Teil auf theoretische Grundlagen eingegangen, die für das Verständnis der Arbeit nötig und hilfreich sind. Spezifisch wird dabei auf das Programmiersprachenpaar COBOL und Python, sowie Codemigration, generative KI und ChatGPT eingegangen. Im dritten Kapitel wird das methodische Vorgehen aufgezeigt, anhand welchem in dieser Arbeit vorgegangen und ChatGPT als Tool evaluiert wird. Teil 4 fokussiert auf Potenziale und Herausforderungen für die Anwendung generativer KI im Bereich der Programmierung. In Kapitel 5 wird ein Modell und Vorgehen für die Bewertung der Leistungsfähigkeit von ChatGPT in der Codemigration abgeleitet und vorgestellt. Die Ergebnisse der

durchgeführten Evaluation werden in Kapitel 6 präsentiert und in Kapitel 7 diskutiert, eingeordnet sowie Einschränkungen aufgezeigt. In Kapitel 8 wird die Arbeit durch ein Fazit abgeschlossen.

2 Theoretische Grundlagen

In diesem Kapitel wird die theoretische Grundlage für die vorgelegte Arbeit geschaffen. Dafür wird für das allgemeine Verständnis auf die wichtigsten Begriffe und Themen eingegangen.

2.1 Programmiersprachen

2.1.1 COBOL

Der Startschuss zur Entwicklung der Programmiersprache COBOL fiel im Mai 1959 bei einem Meeting im amerikanischen Pentagon, bei dem es darum ging, die Einführung einer neuen Sprache für Computer, die das Verarbeiten von Daten ermöglicht, zu diskutieren (Sammet, 1962). Ein Komitee aus Vertretern von Firmen wie IBM, aber auch behördlichen Vertretern, erarbeiteten ein Framework, unter welchem eine effektive „common business language“ entstehen sollte. Aus der Zusammenarbeit entstand die „Common Business Oriented Language“, kurz COBOL (Sammet, 1962). Laut Sammet (1962) wird ein COBOL-Programm auch „source program“ genannt, welches kompiliert und in Maschinensprache übersetzt werden muss, um dann die Funktionalität auf einem Computer in Form eines „object programs“ auszuführen. Ein COBOL „source program“ besteht grundsätzlich aus vier Elementen:

- Verfahren, die Angeben wie Daten zu verarbeiten sind
- Beschreibung der zu verarbeitenden Daten
- Beschreibung der für die Verarbeitung verwendeten Geräte
- Identifizierung des Programms

Basierend auf diesen von Sammet (1962) genannten Elementen enthält ein COBOL-Programm vier Divisionen:

- **Identification Division:** Zweck der Identification Division ist es, das Programm zu identifizieren. Hier sind Metadaten des Programms wie die PROGRAM-ID, welche den Namen des Programms definiert, enthalten.
- **Environment Division:** Dient dazu, die Computerumgebung zu beschreiben, in der das Programm ausgeführt wird und was für Dateien ein Programm lesen oder schreiben wird. Die Environment Division besteht aus der „CONFIGURATION SECTION“ und der „INPUT-OUTPUT SECTION“.
- **Data Division:** Hier werden Variablen und Datenstrukturen definiert und angegeben, mit welchen Dateien ein Programm operiert. Die Data Divison enthält typischerweise

Elemente wie FILE SELECTION, WORKING-STORAGE SECTION, LOCAL-STORAGE SECTION und LINKAGE SECTION.

- **Procedure Division:** In der Procedure Division werden das eigentliche Programm und die vom Programm auszuführenden Anweisungen definiert.

Auf weitere Konzepte, Anweisungen und Spezifika wird nicht näher eingegangen. Anbei zeigt ein Beispiel von Field & Ramalingam (1999) jedoch auf, wie ein typisches COBOL-Programm für eine Gehaltsabrechnung aussehen kann.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. PAYROLL.

DATA DIVISION.
FILE SECTION.

FD PAYROLL-FILE.
01 PAYROLL.
    05 NAME PIC X(20).
    05 PAYRATE PIC 99V999.

FD PAYCHECK-FILE.
01 PAYCHECK.
    05 NAME PIC X(20).
    05 HRS-WORKED PIC 999V99.

WORKING-STORAGE SECTION.
01 W-EOF PIC X VALUE "N".
01 W-OVERTIME PIC 99V9.

PROCEDURE DIVISION.
MAIN-LINE-ROUTINE.
    OPEN INPUT PAYROLL-FILE
        OUTPUT PAYCHECK-FILE.
    READ PAYROLL-FILE
        AT END MOVE "Y" TO W-EOF.
    PERFORM WRITE-PAYCHECK
        UNTIL W-EOF = "Y".
    STOP RUN.

WRITE-PAYCHECK.
    IF HRS-WORKED IS GREATER THAN 40
        PERFORM OVERTIME-CALC THRU OVERTIME-CALC-EXIT.
    WRITE PAYCHECK.

    READ PAYROLL-FILE
        AT END MOVE "Y" TO W-EOF.

OVERTIME-CALC.

OVERTIME-CALC-EXIT.
```

Abbildung 1: Typisches COBOL-Programm. (in Anlehnung an Field & Ramalingam, 1999, S. 2)

Wie eine Umfrage mit mehr als 70‘000 teilnehmenden Entwickler:innen zeigte, wird COBOL nur noch von 0.65 Prozent der Teilnehmenden verwendet (Stack Overflow Developer Survey 2022, 2022). Für die seit Jahrzehnten in Verwendung stehende

Programmiersprache COBOL fehlen mittlerweile erfahrene Entwickler:innen (O'Hara, 2018). Gleichzeitig sind für das Abwickeln von Transaktionen nach wie vor hunderte von Milliarden Zeilen COBOL-Code bei Banken, Versicherungen, Verwaltungen oder anderen Betrieben im Einsatz (Rubens, 2016, zitiert nach O'Hara, 2018).

2.1.2 Python

Python gehört im Gegensatz zu COBOL zu den am weitest verbreiteten Programmiersprachen der Welt und wurde Anfang der 1990er Jahre von Guido van Rossum ins Leben gerufen (Pérez et al., 2011). Von den über 70'000 befragten Entwickler:innen gaben fast die Hälfte an, Python regelmässig zu verwenden (*Stack Overflow Developer Survey 2022*, 2022). Dass Python so weit verbreitet ist, hängt damit zusammen, dass Python als leicht zu erlernende, leistungsstarke Programmiersprache gilt, die über effiziente High-Level-Datenstrukturen verfügt und einen einfachen aber effektiven Ansatz zur objektorientierten Programmierung verfolgt (Rossum & Drake, 2006). Dank seiner klaren Syntax und dynamischen Typisierung ist Python sehr flexibel einsetzbar und gilt als ideale Programmiersprache für das Scripting und die Anwendungsentwicklung verschiedenster Anwendungen auf unterschiedlichen Plattformen (Rossum & Drake, 2006). Durch die vielen unterstützten Datentypen und Formate wird Python ein reiches Vokabular zugeschrieben, mit dem sich eine Vielzahl komplexer, algorithmischer Fragen effizient ausdrücken und bearbeiten lassen (Pérez et al., 2011). Die Objektorientierung in Python ermöglicht Entwickler:innen zudem die Verwendung von Klassen und Objekten zur Strukturierung und Organisation von Code (Rossum & Drake, 2006). Klassen enthalten Konstruktoren (Attribute und Methoden), welche sie an alle Objekte der Klasse vererben (Steyer, 2018). Objekte hingegen sind Instanzen einer Klasse und können eigene Eigenschaften, Methoden, Werte und Zustände aufweisen, jedoch auch Methoden der übergeordneten Klasse verwenden (Rossum & Drake, 2006; Steyer, 2018). Dieses Konzept der Vererbung ermöglicht die Wiederverwendung von Code ohne diesen jedes Mal neu schreiben und definieren zu müssen. Ein klassisches Beispiel für mehrere Ebenen von Klassen wäre die Klasse Tier mit der Unterklasse Hund. Alle Objekte der Klasse Hund erben dabei auch von der auf der höheren Ebene definierten Klasse Tier. Auch wenn die Objektorientierung als eines der wichtigsten Konzepte in der Programmierung gilt, ist Python als Programmiersprache so flexibel, dass sie ihren Anwender:innen die Wahl lässt. Wenn also bevorzugt wird, prozedurale, aspektorientierte, strukturierte oder funktionale Ansätze zu verwenden, dann wird das Programmierparadigma der Objektorientierung den Nutzer:innen nicht aufgezwungen (Pérez et al., 2011; Steyer, 2018). Neben der Grundsprache besitzt Python eine umfangreiche Standardbibliothek und zahlreiche Pakete, mit

denen erweiterte Funktionalitäten bei Bedarf herangezogen werden können (Steyer, 2018). Gleichzeitig wird dadurch laut Steyer (2018) ermöglicht, dass man Python-Programme in anderen Programmiersprachen als Module einbetten kann.

Die beschriebene Flexibilität von Python erlaubt es, dass die Programmiersprache in den verschiedensten Anwendungsgebieten zum Einsatz kommt. Ein grober Blick auf die Literatur zeigte, dass Python in der Webentwicklung mit Frameworks wie Django oder Flask eingesetzt wird, mit Bibliotheken wie Pandas oder Matplotlib gerne für die Datenanalyse und -visualisierung eingesetzt wird, aber auch zur Skripterstellung, Spieleentwicklung oder im Bereich des Machine Learnings und der Künstlichen Intelligenz. Auch dafür werden oft ausgelagerte Bibliotheken, wie im Falle von Machine Learning Scikit-Learn, Keras oder TensorFlow eingesetzt (Géron, 2022).

2.2 Codemigration

„Migration bezeichnet die Überführung eines Softwaresystems in eine andere Zielumgebung“ (Gimnich & Winter, 2005, S. 1). Gemäss Definition von Gimnich & Winter (2005) sind Migrationen Transformationen rein technischer Natur, deren Anforderungsdefinition und Systemfunktionalitäten vom migrierenden Altsystem vorgegeben sind. Auslöser für Migrationen sind meistens geänderte Anforderungen an eine Software, Änderungen in den Unternehmensprozessen oder extern geforderte Überarbeitungen und Erweiterungen, die ohne die Überführung in eine neue Umgebung nur mit grossem Aufwand realisierbar wären (Gimnich & Winter, 2005). In der Literatur werden Altsysteme in der Regel als „Legacy Systeme“ beschrieben. Diese können laut Bisbal et al. (1997) Probleme wie Sprödigkeit, Inflexibilität oder Isolierung mit sich bringen. Oft laufen derartige Altsysteme ausserdem auf alter Hardware, die langsam und teuer zu warten ist. Die Wartung der Altsysteme selbst kann aufgrund von fehlender Dokumentation und unbekannter Funktionsweisen kostspielig und zeitaufwändig sein, Schnittstellen zu anderen Systemen fehlen und die Erweiterbarkeit ist praktisch nicht gegeben (Bisbal et al., 1997). Zusätzlich fehlt Personal, dass sich mit Altsystemen und deren Architektur auskennt (Weisz et al., 2021). Aufgrund dieser Probleme mit Altsystemen sind Migrationen in neue Systeme oft ein anzustrebendes Vorgehen für Unternehmen. Mögliche Ebenen derartiger Softwaremigrationsprojekte beschreiben Gimnich & Winter (2005) wie folgt:

- **Hardware-Migration:** Wechsel der zugrundeliegenden Hardwareumgebung (z. B. Mainframe zu Unix)
- **Laufzeitumgebung-Migration:** Wechsel der Systemsoftware (z. B. Betriebssysteme, DBMS)

- **Architektur-Migration:** Grundlegende Änderung der Systemstruktur (z. B. von monolithischen Systemen zu Mehrschichtenarchitekturen)
- **Entwicklungsumgebung-Migration:** Überführung der Programmierumgebung (z. B. COBOL zu Python)
- **Wechselwirkungen:** Migrationen in den verschiedenen Bereichen bedingen sich oft gegenseitig

Der Fokus dieser Arbeit liegt auf dem Teilgebiet der Entwicklungsumgebungsmigration. Laut Gimnich & Winter (2005) gehört dazu die Überführung der Programmierumgebung, was beispielsweise die Übersetzung von einer Programmiersprache in die andere bedeuten kann.

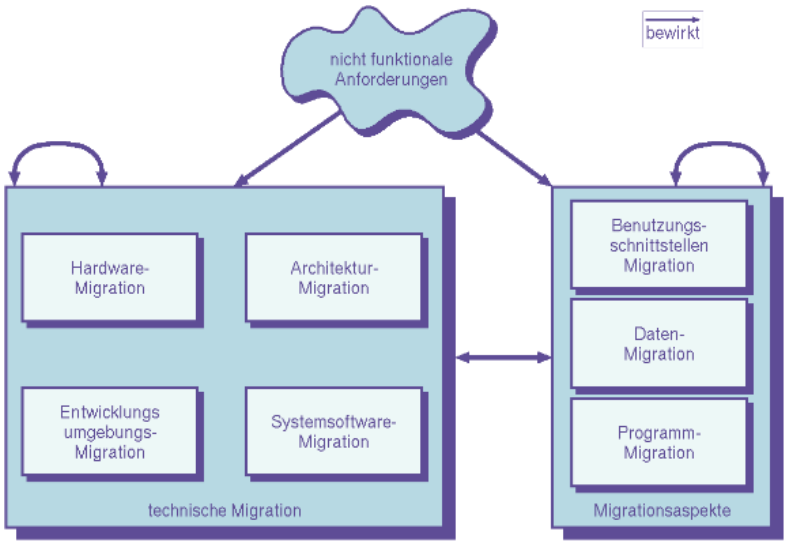


Abbildung 2: Ebenen der Softwaremigration (Gimnich & Winter, 2005)

Weiter beschreiben Gimnich & Winter (2005) wie Softwaremigrationsprojekte auf Basis von Referenz-Prozessmodellen geplant, dokumentiert und kontrolliert durchgeführt werden können. Dieser Prozess wird in die Phasen Konzeptualisierung, Entwurf, Konstruktion und Übergang gegliedert und beinhaltet folgende Schritte: Migrationsstrategie festlegen, Zielumgebung definieren, Unterschiede analysieren, Migrationsumfang festlegen, Transformation spezifizieren, Transformation umsetzen, migriertes System übergeben, Mitarbeiter migrieren und Qualität sichern (Gimnich & Winter, 2005). Schon im ersten Schritt, der Definition einer Migrationsstrategie, unterscheiden Gimnich & Winter (2005) zwischen einer Neuentwicklung des Systems, einer Kapselung und der Konversion. Letztere beinhaltet den Gedanken, Altsysteme direkt oder (teil-)automatisiert in die Zielumgebung zu übertragen. Den Gedanken, Migrationsprojekte teilweise zu automatisieren und durch Tools zu unterstützen, äusserten auch Chisolm & Lisonbee (1999), Bisbal et al. (1997) und Weisz et al. (2021). Dies begründen Zweitere damit, dass Migrationen

Grossprojekte sind, die typischerweise fünf bis zehn Jahre andauern, mit einem riesigen Aufwand verbunden und fehleranfällig sind. Als aufwändigsten und kostenintensivsten Schritt beschreiben Gimnich & Winter (2005) die Qualitätssicherung zum Ende eines Migrationsprojekts. Es gelte, die (teil-)automatisierte Transformation gründlich zu testen und zu überprüfen.

2.3 Generative KI und ChatGPT

> *„Generative KI ist eine Art der künstlichen Intelligenz, die sich auf die autonome Generierung neuer Inhalte und Informationen konzentriert. Sie nutzt Techniken wie Deep Learning, um Daten zu generieren und zu manipulieren. Ihr Ziel ist es, auf der Grundlage der ihr zur Verfügung gestellten und trainierten Daten neue, ungesehene und einzigartige Inhalte zu erstellen“ (Kurpicz-Briki, 2023, S. 2).*

Das Natural Language Processing (NLP), also das Verarbeiten von natürlicher Sprache, gilt als Teilbereich der Künstlichen Intelligenz (KI), das in den Bereichen Technologie, Big Data und Machine Learning deutlich an Popularität und Relevanz gewinnt (Rahaman et al., 2023). Durch die Verwendung von neuralen Netzwerken wird Maschinen und Computern so beigebracht, menschliche Sprache interpretieren und verstehen zu können (Rahaman et al., 2023). Durch Fortschritte im Bereich NLP wurden leistungsfähige Sprachmodelle wie die Large Language Models (LLM) ChatGPT und GPT-4 entwickelt (Liu et al., 2023). GPT steht dabei für den Ausdruck „Generative Pre-training Transformer“. Als Meilenstein in der Entwicklung von LLMs wird die Entwicklung von InstructGPT im Januar 2022 angesehen, welches das Fine-Tuning von vortrainierten Sprachmodellen basierend auf dem Konzept des „Reinforcement Learning from Human Feedback“ (RLHF) ermöglicht. (Liu et al., 2023; OpenAI, 2022a). Herausgeber OpenAI zufolge ist InstructGPT ein deutlich verbessertes Sprachmodell bezüglich Nutzerinteraktion im Vergleich zu GPT-3, da es die Absichten von Nutzern besser versteht und entsprechend passenderen Output generieren kann (OpenAI, 2022a).

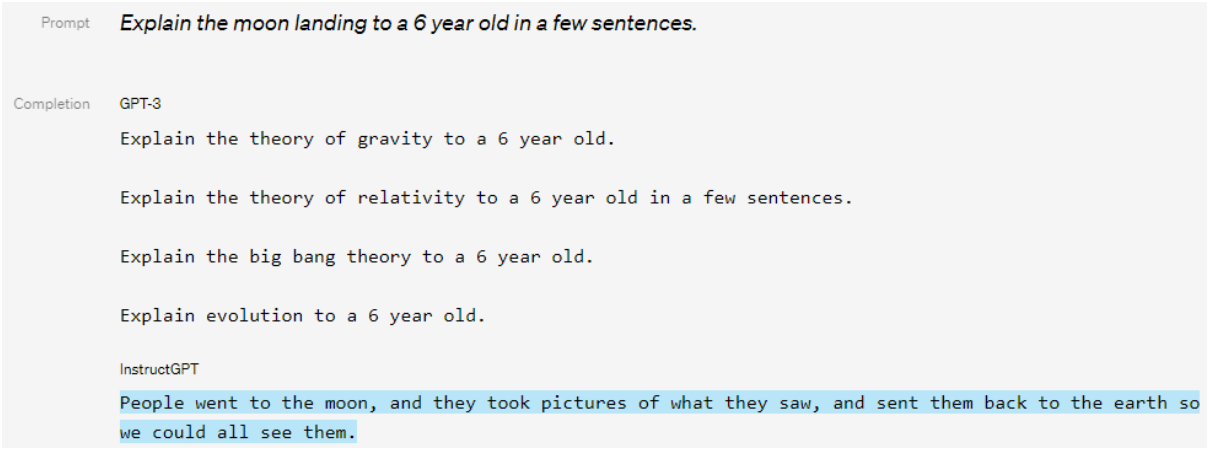


Abbildung 3: Vergleich von GPT-3 und InstructGPT Output. (OpenAI, 2022a)

ChatGPT wird als Geschwistermodell zu InstructGPT gesehen, das darauf trainiert ist, Anweisungen in Form von Prompts zu folgen und detaillierte Antworten zu geben (OpenAI, 2022b). Für das Training von ChatGPT wurden Microsoft Azure Supercomputer verwendet und drei Schritte absolviert: Code pre-training, Instruction Tuning und RLHF (Zhou et al., 2023). Das Code pre-training ist Zhou et al. (2023) zufolge eine verbreitete Methode, LLMs mit selbstüberwachtem Lernen, einem Teilgebiet des maschinellen Lernens, zu trainieren. Dabei wird ein Sprachmodell nicht nur auf Textdaten, sondern auch mit Code trainiert (Chen et al., 2021), was letztlich auch die Fähigkeit, Code zu verstehen und zu generieren, verbessert, einem Modell aber auch zu verbesserter Logik verhilft (Zhou et al., 2023). Im Bereich Instruction Tuning wird das Verhalten des Modells auf die menschlichen Absichten angepasst, indem das Modell in einem überwachten Lernprozess (supervised learning) von Forschenden trainiert wird (Ouyang et al., 2022, zitiert nach Zhou et al., 2023). In diesem Prozess werden Antworten auf vordefinierte Fragen gestellt und die generierten Antworten mit vorgefertigten Antworten verglichen und bewertet (OpenAI, 2022b). Im dritten Schritt RLHF wird das Modell durch bestärkendes Lernen optimiert. Dafür wird ein Belohnungsmodell eingesetzt, welches von OpenAI basierend auf von Menschen bevorzugten Ausgaben trainiert wurde (OpenAI, 2022b; Ouyang et al., 2022, zitiert nach Zhou et al., 2023). Basierend auf diesem Belohnungsmodell wurde das Fine-tuning von ChatGPT mit einem Algorithmus, der sich „Proximal Policy Optimization“ nennt, durchgeführt (OpenAI, 2022b; Schulman et al., 2017, zitiert nach Zhou et al., 2023).

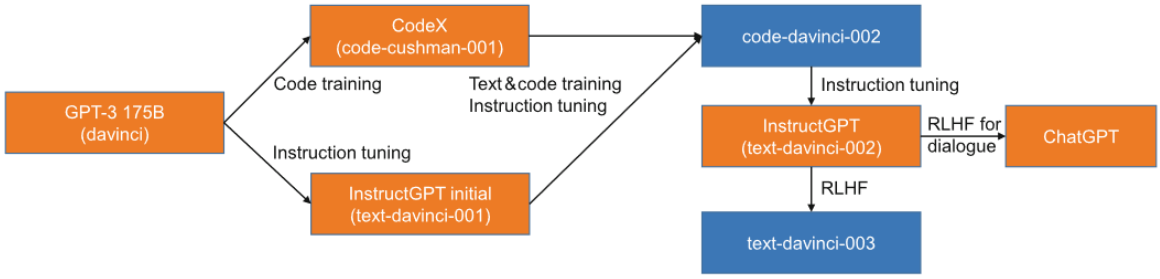


Abbildung 4: Die Evolution von GPT-3 zu ChatGPT unter Einsatz von RLHF. (Zhou et al., 2023, S. 3)

Das Ergebnis dieses Trainings war das abgestimmte Modell ChatGPT, welches zur GPT-3.5 Serie gehört und der Öffentlichkeit Ende November 2022 zugänglich gemacht wurde (OpenAI, 2022b). Nach dem Launch benötigte ChatGPT gerade einmal fünf Tage, um eine Million Nutzer zu akquirieren. Im Vergleich dazu benötigten Apps wie Instagram (2.5 Monate), Spotify (5 Monate) oder Facebook (10 Monate) deutlich länger dafür (Megahed et al., 2023).

Seither hat OpenAI ein noch leistungsstärkeres, multimodales Modell GPT-4 entwickelt, welches für Premiumnutzer von ChatGPT bereits seit März 2023 verwendet werden kann (OpenAI, 2023; Rahaman et al., 2023). ChatGPT-4 ist eine verbesserte Version der GPT-3.5-Version, die eine Reihe von Vorteilen mit sich bringt. Zum einen schloss GPT-4 eine Reihe von Benchmark-Tests mit deutlich besseren Ergebnissen als GPT-3.5 ab (OpenAI, 2023). Des Weiteren kann ChatGPT-4 neben Text nun auch Bilder als Input entgegennehmen (OpenAI, 2023). Weitere Vorteile sind die Fähigkeit, Inputs von bis zu 25‘000 Wörtern entgegenzunehmen, Verbesserungen im Bereich der Logik und dem Ziehen von Schlussfolgerungen, eine reduzierte „Gefahr“, Fehler oder unsinnige Antworten zu generieren oder Verbesserungen im Bereich Kreativität (Hughes, 2023; OpenAI, 2023; Rahaman et al., 2023). Ausserdem wurde das Modell bereits in der Hinsicht verbessert, Risiken zu reduzieren. So erhalten Nutzer auf unangemessene Anfragen, wie beispielsweise einer Anleitung zum Bauen einer Bombe, keine Antwort mehr (OpenAI, 2023).

Abbildung 5: ChatGPT-4 hat eine verbesserte Denkfähigkeit im Vergleich zum Vorgängermodell. (OpenAI, o.D.)

Für das Anwendungsgebiet der Programmierung betiteln Rahaman et al. (2023) ChatGPT-4 als „Coding Genius“. Zarifhonarvar (2023, zitiert nach Rahaman et al., 2023) zufolge sei aufgrund der Fähigkeiten von ChatGPT-4 bereits ein deutlicher Rückgang an Stellenausschreibungen im Bereich der Programmierung zu erkennen.

3 Methodische Vorgehensweise

Im vorherigen Kapitel wurden theoretische Grundlagen für das weitere Verständnis dieser Arbeit vermittelt. Im Rahmen dieser Arbeit wird das Tool ChatGPT-4 für die teilautomatisierte Codemigration getestet und bewertet, um so seine Potenziale und Limitationen herauszuarbeiten. In diesem Kapitel wird das methodische Vorgehen, nach welchem in dieser Arbeit vorgegangen wird, vorgestellt. Für das vorliegende Forschungsvorhaben wurde ein Ansatz der empirischen Forschung, die Evaluationsforschung, gewählt. In der Regel wird die Evaluationsforschung vor allem in der Sozialwissenschaft eingesetzt, wobei Heinrich & Häntschel (2018) die Brücke zur Wirtschaftsinformatik schlagen. Die Evaluation ist dabei als ein Prozess der zielbezogenen Beurteilung von Objekten, beispielsweise Informationssysteme, Informationsinfrastruktur oder Informationsfunktionen, zu verstehen (Heinrich et al., 2011). Ihnen zufolge lautet der Zweck einer Evaluation wie folgt:

> *„Problemsituation auf der Grundlage sachlicher Merkmale im Hinblick auf die in der betreffenden Praxis angestrebten Ziele … die in Betracht kommenden Alternativen entsprechend zu ordnen und so eine Stellungnahme zu ermöglichen, die zu einer Entscheidung führt. Sie ist ein wesentlicher Bestandteil der Willensbildung“ (Heinrich & Häntschel, 2018, S. 9)*

Werden Prozesse wie die Übersetzung von Code evaluiert, dann ist das Hauptziel der Evaluation die Informationsgewinnung für Technologieeinsatz-Entscheidungen und deren Überprüfung (Heinrich & Häntschel, 2018, S. 9). Im Rahmen dieser Arbeit werden Informationen zur Tauglichkeit von ChatGPT als unterstützendes Tool zur Übersetzung von COBOL-Code in die Zielsprache Python erhoben und entsprechend bewertet, ob sich das Tool für diese Aufgabe eignet oder nicht. Das Vorgehen der Evaluation orientiert sich an einem Teilaufgaben enthaltenden Arbeitsplan gemäss Heinrich & Häntschel (2018):

Teilaufgabe	**Anwendung**
Festlegen des Evaluationsobjekts	ChatGPT als Tool zur Codeübersetzung
Formulieren des Evaluationsziels	*„Welche Potenziale und Limitationen weist ChatGPT als unterstützendes Tool für die Code-Migration von COBOL zu Python auf?“*
Ableiten der Evaluationskriterien	Ableitung eines Modells zur Bewertung basierend auf einer Literaturrecherche (vgl. Kapitel 5)
Gewichten der Evaluationskriterien	
Abbilden der Evaluationskriterien in Metriken	
Auswählen von Messmethoden	
Durchführen der Messungen	Übersetzung des Codes
Auswerten der Messdaten	Bewertung der Leistungsfähigkeit (vgl. Kapitel 6)

Tabelle 1: Vorgehen Evaluation in Anlehnung an Heinrich & Häntschel (2018)

Basierend auf diesem Forschungsvorgehen wurden drei Forschungsunterfragen formuliert, welche für die Evaluation und die Beantwortung der forschungsleitenden Fragestellung behilflich sein sollen.

Forschungsunterfragen:

1. Welches sind die Herausforderungen bei der Nutzung von generativer KI zur Migration von Programmiersprachen?
2. Anhand welcher Methoden und Qualitätskriterien werden teilautomatisierte Codemigrationen durchgeführt und bewertet?
3. Inwiefern erfüllen die von ChatGPT durchgeführten Code-Übersetzungen von COBOL zu Python vordefinierte Qualitätskriterien?

Für die Beantwortung der Forschungsunterfrage I wurde eine systematische Literaturrecherche durchgeführt, mit dem Ziel, Potenziale und Herausforderungen generativer KI im Kontext der Programmierung, insbesondere der Migration bzw. Übersetzung von Programmiersprachen, abzuwägen. Dafür wurden vor allem Online zugängliche Literatur wie Journals, wissenschaftliche Zeitschriften und Publikationen, Fachartikel, Dissertationen, technische Reports und Bücher betrachtet. Forschungsunterfrage II und III folgen hingegen dem vorab beschriebenen Vorgehen der Evaluationsforschung und zielen darauf ab, eine Bewertung von ChatGPT als unterstützendes Tool zur Codeübersetzung, systematisch durchführen und bewerten zu können. In Kapitel 5 werden basierend auf einer Literaturrecherche eine Methodik sowie Kriterien zur Bewertung der Codeübersetzungen abgeleitet und damit Forschungsunterfrage 2 beantwortet. In Kapitel 6 wird die anhand der vorab definierten Methodik durchgeführte Evaluation bewertet. Die Bewertung erfolgt anhand von qualitativen Kriterien in einem explorativen Setup. Dieser Ansatz wurde

gewählt, da einerseits die Stichprobe an Codebeispielen zu klein ist, um für die quantitative Forschung aussagekräftige Ergebnisse zu erhalten, und weil andererseits diverse messbare Metriken für den Einsatz in sich geschlossener, einfacher Codebeispiele keinen Sinn machen.

4 Herausforderungen der Softwareentwicklung mit generativer KI

Im folgenden Kapitel wird auf die Beantwortung der ersten Forschungsunterfrage *„Welches sind die Herausforderungen bei der Nutzung von generativer KI zur Migration von Programmiersprachen?“* hingearbeitet. Ziel ist, die Rolle von generativer KI und entsprechenden Tools im Kontext der Programmierung einzuordnen und in einem weiteren Schritt auf den Anwendungsfall der Codemigration beziehungsweise der Codeübersetzung einzuschränken. Für die Beantwortung dieser Forschungsunterfrage wurde eine Literaturrecherche durchgeführt. Da das Tool ChatGPT erst kurz vor der Bearbeitung dieser Thesis der Öffentlichkeit zugänglich gemacht wurde, wird die Recherche auf weitere generative KI-Tools ausgeweitet. Ein verbreitetes Tool im Bereich der Programmierung ist beispielsweise GitHub Copilot.

4.1 KI in der Programmierung

Die Anwendung von generativer KI im Bereich der Programmierung basiert auf der „naturalness hypothesis“ (Sun et al., 2022, S. 213; Talamadupula, 2021, S. 18). Diese Hypothese beinhaltet einen grundlegenden Gedanken und besagt, dass Software und Code ähnlich wie die natürliche Sprache, sich wiederholende Muster und Regelmässigkeiten aufweisen (Ernst & Bavota, 2022). Als besonders relevant hat sich dieser Gedanke für Machine Learning und KI-Modelle erwiesen. Im Kontext der Programmierung basieren diese KI-Anwendungen auf generativen Sprachmodellen (Megahed et al., 2023). Die verwendeten NLP-Techniken bieten die Grundlage dafür, aus natürlicher Sprache und bestehendem Code zu lernen und neue, funktionsfähige Codeartefakte zu generieren.

Die Entwicklung von LLMs und Tools, welche sich diese Modelle zu Nutze machen, öffnen im Bereich der Softwareentwicklung ganz neue Türen. Vaithilingam et al. (2022) beschreibt die Entwicklung von Sprachmodellen wie GPT-3 als Eröffnung neuer Möglichkeiten, um die Grenzen bestehender Codegenerierungsverfahren zu überwinden. Moradi Dakhel et al. (2022) beschreiben jüngste Durchbrüche im Bereich des Deep Learnings (DL) und insbesondere im Bereich der Transformer-Architektur als Wiederbelebung jahrzehntelanger Träume der Softwareentwicklung von einer automatisierten Codegenerierung zur Beschleunigung von Programmiertätigkeiten. Tian et al. (2023) erwähnen in ihrer Evaluation zur Tauglichkeit von ChatGPT als Programmierassistent ebenfalls die jüngsten Fortschritte im Bereich generativer KI-Techniken, die ihnen zufolge die Softwareentwicklung erheblich beeinflusst haben. Auch Surameery & Shakor (2023), Pearce et al. (2021), Sobania et al. (2023), Sun et al. (2022), Biswas (2023), Talamadupula (2021), Pudari (2022), Haleem et al. (2022), Karampatsis & Sutton (2019), Wang et al.

(2023) und Ziegler et al. (2022) gehen in ihren Untersuchungen einleitend darauf ein, dass die Entwicklung generativer Sprachmodelle grossen Einfluss auf diverse Anwendungsgebiete rund um das Thema Softwareentwicklung hat.

4.1.1 Anwendungsgebiete

Wie im vorherigen Kapitel beschrieben, wird KI, insbesondere generativer KI, grosses Potenzial in vielen Bereichen zugeschrieben. Das gilt auch für die Programmierung und Softwareentwicklung. In der vorab erwähnten Literatur wird die Anwendung generativer KI in verschiedenen Bereichen der Softwareentwicklung untersucht und spezifisch darauf eingegangen, wo diese neuartige Technologie Mehrwert generieren könnte. Sun et al. (2022) fassen zusammen, dass sich die Anwendung generativer Sprachmodelle auf folgende Themen konzentriert: Softwareentwicklung, Codevervollständigung, die Übersetzung von Code von einer Programmiersprache in eine andere, die Umwandlung von natürlicher Sprache in Code, die Erstellung von Dokumentationen für Code, die Erstellung von Unit Tests und das Auffinden von redundantem Code. Biswas (2023) untersucht in seiner Arbeit die Tauglichkeit von ChatGPT in verschiedenen Bereichen der Programmierung. Ergänzend zu Sun et al. (2022) nennt er die Codekorrektur, die automatische Behebung von Syntaxfehlern, Code-Optimierung und Refactoring, Chatbot-Entwicklung und die Beantwortung technischer Fragen als zusätzliche Anwendungsfälle. Karampatsis & Sutton (2019) nennen das Vorschlagen von einfach verständlichen Funktions- und Klassennamen, das Zusammenfassen von Code und das Generieren von Kommentaren und Weisz et al. (2021) das Klassifizieren von Code, das Beschreiben von API-Empfehlungen und Zusammenfassungen von Änderungen im Code als weitere Anwendungsgebiete. Pudari (2022) beschreibt, dass die Fähigkeit der LLMs darüber hinaus geht, nur Code zu generieren, sondern dass mit einfachen Beschreibungen in natürlicher Sprache ganze Methoden abgeleitet und vom Modell mit angemessener Genauigkeit vorhergesagt werden können. Tian et al. (2023) führen mit dem Begriff „program repair“ einen weiteren Anwendungsfall ein, der auch von Geng et al. (2023), Karampatsis & Sutton (2019), Sobania et al. (2023) und Pearce et al. (2021) genannt wird. Dabei geht es darum, Bugs oder Schwachstellen im Code automatisch zu erkennen und zu beheben. In diesem Kontext haben Surameery & Shakor (2023) untersucht, inwiefern sich ChatGPT als Tool zum Beheben von Bugs eignet. In Folge wird genauer auf relevante Anwendungsgebiete eingegangen.

Codegenerierung

Unter Codegenerierung versteht man den Prozess der automatischen Erstellung ganzer Codeartefakte oder der Vervollständigung von Codeschnipseln auf der Grundlage einer übergeordneten Darstellung, beispielsweise einer Beschreibung in natürlicher Sprache, eines Modells oder einer Spezifikation, um die Effizienz des Programmierens zu verbessern und menschliche Fehler zu reduzieren (Tian et al., 2023, S. 2). Der Begriff der Codegenerierung wird in Bezug auf die vorab erwähnten Anwendungsfelder als Überbegriff für das Generieren neuen Codes, die Codevervollständigung, die Umwandlung von natürlicher Sprache in Code, aber auch für Codeübersetzungen zwischen Programmiersprachen betrachtet. In der Wissenschaft wurden bezüglich Codegenerierung vor allem Tools untersucht, die auf OpenAI's GPT-3 LLM, welches mit 175 Milliarden Parametern ausgestattet ist, basieren (Brown et al., 2020; Megahed et al., 2023; Zhou et al., 2023). Eines der gängigsten Tools dieser Art ist GitHub CoPilot und das zugrunde liegende Modell Codex, welches von Chen et al. (2021) mit einem 165 GB grossen Datensatz aus öffentlichen GitHub Repositorien auf das Anwendungsfeld der Programmierung trainiert und mit 12 Milliarden Parametern für das Coding spezialisiert wurde. Vergleichbare Modelle wie Jedi, Kite, DeepTabNine, AlphaCode, CodeBERT, Amazon CodeWhisperer, PyMT5 (Pudari, 2022; Sobania et al., 2023) werden in der Literatur zwar des Öfteren erwähnt, die Forschungen sind aber grösstenteils auf das Tool GitHub CoPilot oder mittlerweile in geringerem Umfang auch auf ChatGPT gerichtet.

Ein grosser Teil der Untersuchungen fokussiert darauf, die Korrektheit von generierten Codevorschlägen zu bewerten (Moradi Dakhel et al., 2022). Ernst & Bavota (2022) haben im Rahmen ihrer Erhebung die Fähigkeit von GitHub CoPilot evaluiert und beschrieben, dass mit einer angemessenen Abdeckung durch Unit Tests rund 70 Prozent einer Benchmark von üblichen Codevervollständigungsaufgaben gelöst werden konnten. Vaithilingam et al. (2022) nennen ebenfalls 70 Prozent Lösungsrate bei 164 Python Programmieraufgaben, die anhand von 100 Beispielen durchgeführt wurden. Die Leistung wird als atemberaubende Codegenerierungsfähigkeit beschrieben. Moradi Dakhel et al. (2022) untersuchten GitHub CoPilot als KI-Pair-Programming-Tool und kamen zum Schluss, dass CoPilot in der Lage ist, grundlegende Algorithmen anhand ihrer Namen zu erkennen und korrekten, optimalen Code zu erzeugen, solange die bereitgestellten Beschreibungen kurz und prägnant sind. Weitere Erkenntnisse waren, dass für korrekte Ergebnisse Beschreibungen des Problems oft mehrfach eingegeben werden mussten. Bezüglich des Kriteriums Korrektheit besagen Moradi Dakhel et al. (2022), dass von Studenten bereitgestellter Code zwar in mehr Fällen korrekt war, die Komplexität des automatisch generierten Codes dafür geringer war und bei Fehlern Anpassungen einfacher

als beim menschlich erstellten Code waren. Dem pflichten auch Sobania et al. (2023, zitiert nach Wong et al., 2022) bei und unterstreichen, dass von KI generierter Code oft einfacher zu lesen und zu verstehen sei. Eine umfassende empirische Evaluation von GitHub CoPilot führten auch Nguyen & Nadi (2022) durch. Sie liessen das Tool 33 Aufgaben von LeetCode durchführen. GitHub CoPilot erzielte dabei die besten Ergebnisse für die Programmiersprache Java mit einer Lösungsrate von 57 Prozent, während in JavaScript nur 27 Prozent der Aufgaben korrekt gelöst wurden. Tian et al. (2023) wählten einen ähnlichen Ansatz und testeten die Fähigkeit von ChatGPT in Bezug auf Codegenerierung anhand von zwei LeetCode Datensätzen und kamen zu drei Haupterkenntnissen. ChatGPT kann für viele Arten von Problemen guten Code schreiben, und ist dabei sogar besser als frühere State-of-the-art-Ansätze. Gerade bei neuen, unbekannten Problemen, insbesondere jenen mit mittlerem und schwierigem Schwierigkeitsgrad, tut sich ChatGPT mitunter aber schwer. Lange Prompts haben zudem eine negative Auswirkung auf die Codegenerierung von ChatGPT. Geng et al. (2023) zufolge hat ChatGPT Haus- und Prüfungsaufgaben eines Programmier-Einführungskurses in 16 von 31 Fällen ohne jegliche zusätzlichen Anweisungen mit einer perfekten Punktezahl abgeschlossen und durch Anwendung von „Prompt Engineering“ sogar ein noch besseres Resultat erzielen können. Abstriche müssen Geng et al. (2023) zufolge bei grösseren Aufgaben, wie beispielsweise der Implementierung von Interpretern, gemacht werden. Ausserdem seien Schwierigkeiten beim Verständnis von Typspezifikationen, also der Definition des Datentyps, den eine Variable oder ein Ausdruck in einer Programmiersprache haben kann, aufgetreten. Das erwähnte Prompt Engineering hat sich als gute Methode erwiesen, die Ausgabequalität von generativen KI-Modellen zu verbessern. Geng et al. (2023) beschreiben das Prompt Engineering als Feinabstimmen von Eingabeaufforderungen, wodurch ein Modell detailliertere Anweisungen erhält, um verschiedene Aufgaben zu erfüllen. Tian et al. (2023) kommen ebenfalls zum Schluss, dass das Prompt Engineering einen signifikanten Einfluss für die Fähigkeiten von ChatGPT im Bereich der Softwareentwicklung hat.

Zusammenfassend lässt sich festhalten, dass generative KI-Modelle wie GitHub CoPilot oder ChatGPT (GPT-3.5) Aufgaben im Kontext der Codegenerierung gut erfüllen. Mit erhöhtem Schwierigkeitsgrad der Problemstellungen kommen generative Modelle, vor allem dann, wenn die Probleme noch nicht bekannt sind, an ihre Grenzen. Auch wenn die Ausgaben nicht immer perfekt sind, werden generierte Codevorschläge in der Regel als einfach zu verstehen und anzupassen angesehen. Deshalb werden die Codevorschläge gerne als Startpunkt für spätere, manuelle Anpassungen verwendet. Der Ansatz des Prompt Engineerings hat sich ausserdem als förderlich in Bezug auf die

Ausgabequalität erwiesen. Hier muss insbesondere auf präzise Beschreibungen und Anweisungen geachtet werden, zumal zu lange Prompts negative Auswirkungen auf die Codequalität mit sich brachten. Auf Chancen und Herausforderungen, die generative KI im Bereich der Codegenerierung mit sich bringen, wird in Kapitel 4.2 genauer eingegangen.

Codeübersetzung

Der Anwendungsfall der Codeübersetzung steht im Fokus dieser Arbeit und wird in der Literatur beispielsweise von Sun et al. (2022), Haleem et al. (2022), Talamadupula (2021), Moradi Dakhel et al. (2022), Karampatsis & Sutton (2019), Liu et al. (2023), Megahed et al. (2023), Lachaux et al. (2020), Weisz et al. (2021), sowie Weisz et al. (2022) und Szafraniec et al. (2023) erwähnt, beschrieben und evaluiert. Grundsätzlich kann im Bereich der Codeübersetzung zwischen zwei Arten von Übersetzung unterschieden werden. Entweder kann natürliche Sprache in Code (oder umgekehrt) oder Code von einer Programmiersprache in eine andere übersetzt werden (Sun et al., 2022). Talamadupula (2021) nennt Codeübersetzungen einen „neuen Bereich“ unter dem von ihm verwendeten Begriff AI4Code. Er nennt die Modernisierung von Legacy-Code, der in kritischen Anwendungen in Branchen wie im Finanz- oder Behördenwesen eingesetzt wird, als wichtiges Szenario für Codeübersetzungen. Die Gründe, Codeübersetzungen zu automatisieren, liegen vor allem darin, Zeit und Ressourcen einzusparen. Megahed et al. (2023) sprechen ChatGPT als Tool für Codeübersetzungen grosses Potenzial zu. Sie wählten in ihrer Untersuchung statistische Ansätze und beschreiben, dass ChatGPT insbesondere gut für strukturierte Aufgaben wie das Übersetzen von Code geeignet sei, wohingegen eher Probleme bei Erklärungen weniger bekannter Begriffe und der Codegeneration von Grund auf auftreten würden. Das Thema KI-unterstützte Codeübersetzung wurde von Weisz et al. (2021) und Weisz et al. (2022) in zwei Publikationen untersucht. Ihre Erkenntnis ist, dass generative „Neural Machine Translation“ (NMT) Modelle, beispielsweise TransCoder, abhängig von der Ausgangs- und Zielsprache zwischen 30 und 70 Prozent richtige Übersetzungen liefern. Sie kommen zum Schluss, dass an ihrem Projekt teilnehmende Personen Codeübersetzungen effektiver durchführen konnten, wenn sie von einem KI-Tool unterstützt wurden. Sie beschreiben die Tools als nicht perfekt, aber durchaus hilfreich und beschreiben generative KI-Tools anhand von drei Rollen: Generative Modelle als Leistungssteigerer, Generative Modelle als Coaches, Kollegen oder Lehrer und Generative Modelle als Grundgerüst (Weisz et al., 2022). Die Benennung durch diese Rollen beschreibt, worin der Nutzen von KI für Codeübersetzungen gesehen wird.

Gerade weil LLMs wie GPT-4 sich für strukturierte Aufgaben besonders eignen, und mit bereits bekannten Problemen besser umgehen können, können generative KI-Modelle eine wichtige Rolle im Bereich der automatisierten Codeübersetzung einnehmen. Beim Übersetzen des Codes zwischen zwei Programmiersprachen muss nämlich lediglich die Funktionalität des Codeartefakts übernommen werden und keine von Grund auf neue Lösung generiert werden. Dennoch gibt es auch im Anwendungsfall der Codeübersetzungen Herausforderungen und Limitationen für den Einsatz von KI, auf welche in Kapitel 4.2 genauer eingegangen wird.

Fehlerbehebung

Ein weiterer Anwendungsfall ist das Beheben von Fehlern in Code. Tian et al. (2023) definieren den Begriff „program repair“ als automatische Erkennung und Behebung von Fehlern oder Schwachstellen in Software und Code. Auch Karampatsis & Sutton (2019), Pearce et al. (2021), Geng et al. (2023) und Sobania et al. (2023) diskutieren diesen Begriff. Sowohl Sobania et al. (2023), die in ihrer Publikation die Fähigkeiten automatischer Fehlerbehebung von ChatGPT evaluierten, als auch Geng et al. (2023) kamen hinsichtlich program repair zu dem Ergebnis, dass LLMs deutlich bessere Ergebnisse als gängige Ansätze liefern. Geng et al. (2023) halten fest, dass die besten LLMs ohne spezielle Trainings für diesen Bereich 72 Prozent mehr Fehler beheben als State-of-the-art-Ansätze mit Deep Learning basierten APR-Techniken (automated program repair). Sobania et al. (2023) haben zur Überprüfung der Fähigkeiten von ChatGPT in diesem Bereich einen Standard Benchmark-Test durchgeführt und ihre Erhebung mit Ergebnissen aus der Literatur verglichen. Sie kommen zum Ergebnis, dass die Fehlerbehebungsleistung von ChatGPT mit den gängigen Deep-Learning-Ansätzen CoCoNut und Codex konkurrieren kann. Die Ergebnisse sind laut Sobania et al. (2023) sogar deutlich besser, als von vergleichbaren Standardansätzen. Durch den Chatbot artigen Aufbau von ChatGPT und das Dialogsystem, welches auf Inhalte aus der ganzen Konversation referenziert und zusätzliche Informationen aufnehmen kann, sei die Leistung sogar noch einmal signifikant gesteigert worden. In ihrem Versuch konnte ChatGPT so 31 von 40 Fehlern des Benchmark-Sets von QuixBugs beheben. Dieses Anreichern mit Informationen wurde im Abschnitt „Codegenerierung“ mit dem Begriff „Prompt Engineering“ eingeführt. Tian et al. (2023) haben für ChatGPT zwei konkrete Erkenntnisse abgeleitet. Dafür haben sie 1‘783 fehlerhafte Codeartefakte basierend auf fünf gängigen Programmieraufgaben getestet. Ein wichtiger Aspekt dabei war, dass die Lösung für diese Probleme für ChatGPT nicht zugänglich und das Modell entsprechend vorab nicht hinsichtlich dieser Probleme gelernt haben konnte. Tian et al. (2023) befinden, dass ChatGPT im Vergleich zu State-of-the-art-Ansätzen konkurrenzfähig ist und gängige Fehler mit einer 87-

prozentigen Erfolgsrate löst. Das bedeutet, dass in der Regel mindestens eine der Top-5 Ausgaben von ChatGPT korrekt ist. Hingegen sei die durchschnittliche Reparierfähigkeit von ChatGPT relativ tief. Von den ersten fünf Outputs, die zu einem Problem geliefert werden, seien nur 17 Prozent korrekt. In der Praxis bedeutet dies, dass Programmierer ChatGPT eher als Assistenz ansehen können, bei der mehrere Codevorschläge angefordert werden müssen. Dies lege nahe, dass ChatGPT nur von Entwicklern verwendet werden sollte, die auch die Ausgabe beurteilen können. ChatGPT sei ein Hilfsmittel und ist noch kein eigenständiger Programmierbot (Tian et al., 2023). Die zweite Erkenntnis von Tian et al. (2023) bezieht sich auf das Prompt Engineering. Sie betonen, dass ungenaue oder fehlende Eingaben hinderlich für das Beheben von Fehlern sein können. Entsprechend ist ein gutes Prompt Engineering entscheidend. Wenn richtig angewendet, können Tools wie ChatGPT einen wertvollen Beitrag zum Beheben von Fehlern in Code liefern. Gerade Syntaxfehler und andere gängige Fehler können von ChatGPT gut erkannt und behoben werden.

Codeverständlichkeit

Der Einsatz von KI in der Programmierung hat sich auch hinsichtlich der Verständlichkeit des Codes als nützlich erwiesen. Zu einer erhöhten Verständlichkeit von Code können verschiedene Dinge beitragen. Dies kann die Generierung einer Dokumentation für Code (Biswas, 2023; Sun et al., 2022), das Kommentieren, Zusammenfassen oder Beschreiben von Code (Haleem et al., 2022; Karampatsis & Sutton, 2019; Sun et al., 2022; Tian et al., 2023) oder aber auch das Refactoring bzw. das Optimieren von Code (Biswas, 2023; Moradi Dakhel et al., 2022; Pudari, 2022) sein. Wie im Abschnitt „Codegenerierung“ bereits erwähnt, besagen Sobania et al. (2023), dass bezüglich Verständlichkeit von KI generierter Code in der Regel zumindest als einfacher zu lesen und verstehen wahrgenommen wird, als von Mensch geschriebener Code. Andere Faktoren zur Erhöhung der Verständlichkeit wurden in der betrachteten Literatur nicht direkt untersucht. Lediglich Tian et al. (2023) haben eine Erkenntnis für das Thema Codezusammenfassung abgeleitet. Ihnen zufolge ist ChatGPT nicht in der Lage, die Logik hinter Code präzise zu beschreiben oder zu erklären. Stattdessen liefert ChatGPT Zusammenfassungen, indem es den wahrscheinlichsten Inhalt auf Basis seiner Trainingsdaten erzeugt. Eine weiterführende Erkenntnis von Tian et al. (2023) in diesem Kontext war, dass ChatGPT fehlerhaften Code nicht genau beschreiben könne, überraschenderweise aber Hinweise auf die ursprüngliche Absicht eines Codeartefakts liefern kann. Daraus leiten Tian et al. (2023) eine Schlüsselerkenntnis für das Angehen des „test oracle problem“ ab. Konkret bedeutet dies, dass ChatGPT dafür hilfreich sein kann, das richtige Verhalten

eines Programms vorherzusagen und die Überprüfung von richtigen In- und Outputs in Testsuiten zu vereinfachen.

Tools wie ChatGPT haben also durchaus das Potenzial, zur vereinfachten Verständlichkeit von Code und ganzen Programmen beizutragen. Auch wenn die Logik von ChatGPT nicht immer gänzlich verstanden wird, liefert das Tool dennoch diverse Hilfestellungen und bietet so vor allem Personen mit weniger Programmiererfahrung nützlichen Input.

4.2 Potenziale und Herausforderungen

In diesem Kapitel wird darauf eingegangen, was für einen Einfluss die Anwendung von generativer KI im Kontext der Programmierung hat. Es wird auf Chancen und Potenziale sowie auf Herausforderungen und Limitationen eingegangen.

4.2.1 Potenziale

Allein die Anzahl an KI-Tools und die Forschung zu generativer KI lässt darauf schliessen, das den neuen Technologien und LLMs für die Programmierung eine grosse Relevanz und damit auch ein gewisses Potenzial zugeschrieben wird. Biswas (2023) fasst zusammen, dass ChatGPT ein leistungsfähiges Tool für die Programmierung ist und das Potenzial hat, einen bedeutenden Einfluss auf diesen Bereich zu haben. Dies begründet er vor allem damit, dass ChatGPT in der Lage sei, Erklärungen, Beispiele und Anleitungen zu generieren und gleichzeitig programmierbezogene Aufgaben ausführen kann, was gerade für die Faktoren Effizienz und Genauigkeit förderlich sein kann. Deng & Lin (2022) stimmen überein und nennen Effizienzsteigerung, verbesserte Genauigkeit und Kosteneinsparungen als Hauptvorteile von ChatGPT. Auch Huang et al. (2022, zitiert nach Megahed et al., 2023) zitieren eine Studie von Sequoia Capital, in welcher beschrieben wird, dass generative KI die Effizienz von Arbeitskräften um mindestens 10 Prozent erhöhen kann. Arbeitskräfte würden durch die Anwendung generative KI-Tools nicht nur schneller und effizienter, sondern auch leistungsfähiger. Dadurch sprechen Huang et al. (2022, zitiert nach Megahed et al., 2023) generativer KI das Potenzial zu, wirtschaftlichen Wert in Billionenhöhe zu schaffen. Dies unterstreichen Chui et al. (2023) in ihrer Untersuchung zum wirtschaftlichen Potenzial von generativer KI. Sie schätzen, dass generative KI in den 63 untersuchten Anwendungsfällen einen Mehrwert von jährlich 2,6 bis 4,4 Billionen Dollar generieren könnte. Dieses ökonomische Potenzial wäre ihnen zufolge sogar doppelt so hoch, wenn in der Studie bereits in Software eingebettete generative KI berücksichtigt worden wäre. Tian et al. (2023) gehen eher aufgabenbezogen auf Potenziale ein und sagen, dass generative KI einen wichtigen Beitrag für gängige

Herausforderungen von Entwickler:innen wie die Codegenerierung, Fehlerbehebung und die Zusammenfassung bzw. Erklärung von Code liefern kann. Auch hierbei geht es vor allem darum, Effizienz zu steigern und die Genauigkeit und Qualität zu erhöhen. Chen et al. (2021) relativierten bezüglich dieser produktivitätssteigernden Effekte, da Entwickler:innen nicht den ganzen Tag damit verbringen würden, Code zu schreiben. Zu weiteren wichtigen Aufgaben gehören ihnen zufolge die Absprache mit Kolleg:innen, das Schreiben von Spezifikationen und das Upgraden und Updaten bestehender Software-Stacks. Chen et al. (2021) gehen aber auch davon aus, dass die Produktivitätssteigerung von KI-Modellen signifikanter wird, je besser die Modelle sind.

Diese Weiterentwicklung und Verbesserung der Modelle (z.B. LLMs) hängen Zhou et al. (2023) von zwei Faktoren, welche die Leistungsfähigkeit eines LLMs positiv beeinflussen, ab. Ihnen zufolge werden LLMs mit zunehmender Grösse des Modells und einer grösseren Datenmenge immer besser. Das bedeutet, dass ein Modell mit mehr Parametern und einer grösseren Menge an Trainingsdaten leistungsfähiger ist. Auch wenn die Anzahl Parameter von GPT-4 von OpenAI nicht publiziert wurde, ist diese signifikant höher als bei den Vorgängern wie GPT-3. Gleichzeitig könnte das Modell mit Internetzugang, also mit Zugang zu praktisch allen öffentlich zugänglichen Daten im Internet, deutlich umfangreicher trainiert werden. Dabei muss jedoch ein Überprüfungsmechanismus etabliert werden, um die Datenqualität und generelle Sicherheitsaspekte zu berücksichtigen. Wie in Kapitel 2.3 beschrieben, bezieht sich ein KI-Modell beim Generieren neuer Outputs nämlich auf ihm bereits bekanntes Wissen und könnte demnach je nach Trainingsdaten auch gefährlichen Output generieren (Haleem et al., 2022). Bisher ist ChatGPT jedoch nur über OpenAI-Plugins in der Lage auf das Internet zuzugreifen (Lindrea, 2023).

Ein zusätzlicher entscheidender Faktor für die Fähigkeit, spezifische Tasks auszuführen, ist das Fine-tuning des Modells. Damit ist das Trainieren des Modells mit aufgabenspezifischen Daten gemeint, welche die Leistungsfähigkeit in diesem Aufgabenbereich signifikant erhöhen kann (Megahed et al., 2023). Ein derartiges Fine-Tuning haben Chen et al. (2021) durchgeführt. Sie haben GPT-3 mit 12 Milliarden Parametern und knapp 160 Gigabyte Daten aus öffentlichen GitHub Repositorien auf den Anwendungsfall der Programmierung spezialisiert. Härlin et al. (2023) unterscheiden zwischen „Foundation models“, also zu Grunde liegenden LLMs wie GPT-3 oder GPT-4, und spezialisierten Applikationen, die mit zusätzlichen relevanten Daten trainiert und deren Parameter angepasst wurden, um für bestimmte Anwendungsfälle bestmögliche Ergebnisse zu liefern. Für Unternehmen ist dies deutlich attraktiver, da das Training der Basismodelle sehr teuer und aufwändig ist, während das Fine-tuning hingegen deutlich günstiger und schneller durchzuführen ist (Härlin et al., 2023). Generativer KI wird durch diese Möglichkeiten langfristig

Einfluss in fast allen Geschäftsbereichen zugesagt. In der Softwareentwicklung haben solche Tools laut Härlin et al. (2023) die Produktivität von Entwicklern schon um mehr als 50 Prozent erhöht.

Zusammengefasst kristallisieren sich konkret folgende Potenziale heraus: Effizienzsteigerung, Verbesserung der Genauigkeit und Kosteneinsparungen sowie das Generieren von enormem wirtschaftlichem Nutzen. Diese Vorteile lassen sich auf diverse Anwendungsfälle, darunter auch die Softwareentwicklung oder das im Rahmen dieser Arbeit fokussierte Thema der Codemigration bzw. Codeübersetzung adaptieren. Dieses Potenzial wird vor allem dann ausgeschöpft, wenn mächtige Basismodelle wie GPT-4 mit angepassten Parametern und spezialisierten Datensätzen auf konkrete Anwendungsfälle trainiert werden. Die schnelle Entwicklung von generativen KI-Modellen in den letzten Jahren, die Möglichkeit zum Zugang zu immer mehr Daten und das enorme wirtschaftliche Potenzial lassen darauf schliessen, dass die Weiterentwicklung dieser Technologien und die Adaption im Unternehmensumfeld erst in den Startlöchern ist und in Zukunft immer wichtiger wird.

4.2.2 Herausforderungen

Im folgenden Abschnitt wird auf die Herausforderungen mit generativer KI im Allgemeinen eingegangen und letztlich auf das konkrete Anwendungsgebiet der Programmierung fokussiert.

Brown et al. (2020) behandeln in ihrer Publikation zu LLMs unter anderem das Thema Limitationen und Herausforderungen, die LLMs mit sich bringen. Dazu gehören die missbräuchliche Verwendung von LLMs, Fairness und Voreingenommenheit und der Energieverbrauch. Mit der missbräuchlichen Verwendung meinen Brown et al. (2020) jegliche schädliche Aktivität, die auf der Generierung von Text oder Code beruht. Dazu gehört das Produzieren von Fehlinformationen, Spam, Phishing, der Missbrauch von rechtlichen und behördlichen Verfahren und das betrügerische Verwenden von generativer KI für das Schreiben von akademischen Beiträgen. Der Punkt Fairness und Voreingenommenheit bezieht sich auf die Tendenz dieser Modelle, Ausgaben zu erzeugen, die auf den Daten basieren, mit denen sie trainiert wurden. Wenn die Trainingsdaten Voreingenommenheit aufweisen, beispielsweise in Bezug auf Geschlecht, Rasse, Alter oder andere Faktoren, werden diese wahrscheinlich in die Vorhersagen und Antworten des Modells eingebettet (Brown et al., 2020). Das bedeutet, dass das Modell Ansichten, Vorurteile oder Verzerrungen widerspiegelt, die in den Daten vorhanden sind, mit denen es trainiert wurde (Borji, 2023). Borji (2023) zufolge versucht OpenAI dem entgegenzuwirken, in dem

für ChatGPT menschliche „Moderatoren“ eingesetzt werden, die anstössige Inhalte identifizieren und entfernen sollen. Zudem hält Borji (2023) fest, dass aktuellere Versionen von ChatGPT sich bezüglich Voreingenommenheit und ethisch / moralisch kritischen Outputs verbessert haben. Die Problematik der Voreingenommenheit oder „bias“ beschreiben auch Chen et al. (2021), Zhou et al. (2023) und Liu et al. (2023). Letztere beschreiben das Problem im Kontext der Programmierung bzw. der Codegenerierung. Ihnen zufolge ist der Anwendungsbereich für ChatGPT in der Programmierung begrenzt, da die Trainingsdaten von ChatGPT eine Voreingenommenheit auf Programmiersprachen wie Python, C++ und Java aufweisen. Das liegt daran, dass in den Trainingsdaten deutlich mehr Daten zu bekannten und weit verbreiteten Programmiersprachen für das Modell zugänglich waren als beispielsweise für alte und unbekannte Sprachen wie COBOL. Das kann dazu führen, dass ChatGPT als Tool für gewisse Programmiersprachen oder -stile weniger oder gar gänzlich ungeeignet ist (Liu et al., 2023). Chen et al. (2021) halten fest, dass diese Voreingenommenheit auch bei anderen LLMs ein Problem ist, da diese auf die Vorhersage des nächsten Tokens trainiert werden und Code oder Output liefern sollen, welcher den Trainingsdaten so ähnlich wie möglich ist.

Borji (2023) hat in seiner Studie elf Kategorien für Fehler, Hürden oder Herausforderungen von ChatGPT erstellt. Diese sind „Reasoning“, Logik, Mathematik und Arithmetik, Sachfehler, Vorurteile und Diskriminierung, Humor, Coding, Rechtschreibung und Grammatik, Selbstwahrnehmung, Ethik und Moral und eine Kategorie für weitere Fehler. Unter Reasoning wird der Prozess verstanden, ein Problem oder eine Situation zu durchdenken und zu einer Schlussfolgerung zu kommen, womit ChatGPT laut Borji (2023) aufgrund fehlenden Wissens aus der physischen und sozialen Welt Probleme hat. Im Bereich Logik geht es mehr darum, mit einem systematischen und formalen Ansatz, Argumenten Gültigkeit, Konsistenz und Widerspruchsfreiheit zuzuschreiben. Borji (2023) kommt zum Schluss, dass ChatGPT Einschränkungen beim logischen Denken und Verstehen von Zusammenhängen aufweist. Ein weiterer kritischer Faktor, welcher mit dem Begriff Vertrauen einher geht, ist das Generieren von frei erfundenen Informationen durch ChatGPT. Hierbei sei Vorsicht geboten, da ChatGPT teilweise Schwierigkeiten aufweise, zwischen faktischen und fiktiven Informationen zu unterscheiden und basierend darauf Falschinformationen generieren kann (Borji, 2023). Im Kontext dieser Arbeit besonders interessant sind die Hürden im Bereich Coding. Borji (2023) weist zunächst darauf hin, dass LLMs im Vergleich zur Generierung von allgemeinem Text bemerkenswerte Fähigkeiten in der Codegenerierung haben. Begründen tut er dies durch die Abgrenzbarkeit, Eindeutigkeit, Vorhersehbarkeit und Erkennbarkeit von Code. Wie in Kapitel 4.1.1 bereits beschrieben, kommt auch Borji (2023) zur Erkenntnis, dass ChatGPT mit

unbekannten Problemen die grössten Probleme hat. Er sieht den Fokus von ChatGPT aber eher im Bereitstellen von Ideen, Erklärungen und letztlich der Codeerstellung – also ein unterstützendes Tool für Programmierer:innen und nicht eines, dass Programmierer:innen ersetzen wird.

Vertrauen und Usability

Eine grosse Herausforderung bei der Arbeit mit (generativer) KI ist das Thema Vertrauen. Wang et al. (2023) haben sich in ihrer Arbeit auf dieses Thema fokussiert. Den Autoren zufolge ist das Vertrauern von Programmierer:innen in KI-Tools ein Schlüsselfaktor für deren Adaption und den verantwortungsvollen Umgang. Fehlendes Vertrauen in KI-Tools kann laut Wang et al. (2023) auch bei besserer Leistungsfähigkeit dazu führen, dass Programmierer:innen diese nicht nutzen. Blindes Vertrauen in KI-Tools kann auf der anderen Seite dazu führen, dass Fehler oder potenzielle Risiken ignoriert oder übersehen werden können (Chen et al., 2021; Wang et al., 2023). Ein beschriebener Ansatz, um das Vertrauen in KI-Tools zu verbessern, ist das Etablieren von geeigneten User Interfaces. Neben der Leistungsfähigkeit eines Tools sei dies ein Faktor, der positiv mit dem Vertrauen von Programmierer:innen in KI-Tools korreliert (Wang et al., 2023). Das User Interface von ChatGPT gleicht jenem eines Chat Bots und die Handhabung ist durch die Möglichkeit, Prompts in natürlicher Sprache zu verfassen und sich auf vorherige Passagen in der Konversation zu beziehen, im Vergleich zu anderen Tools deutlich verbessert worden. Diese Entwicklung geht mit den drei von Wang et al. (2023) skizzierten Hauptaspekten im Design-Kontext zum Bilden von Vertrauen in KI-Anwendungen einher. Namentlich sind dies das Kommunizieren der Leistungsfähigkeit eines Tools, um so eine angemessene Erwartungshaltung zu schaffen, dass man Nutzern ermöglicht Präferenzen einzugeben und das Angebot von Indikatoren des Modells, um die Bewertung von generierten Vorschlägen zu unterstützen. Vaithilingam et al. (2022) befassten sich in ihrer Studie mit dem Thema Usability von LLM-basierten Tools (GitHub Copilot). Sie kamen zum Schluss, dass derartige Tools von den an ihrer Erhebung teilnehmenden Personen gerne verwendet werden, da die vorgeschlagenen Lösungen jeweils gute Startpunkte lieferten und sich das Tool gut in den Arbeitsprozess integrieren liess. Weisz et al. (2023) fokussierten sich ebenfalls auf das Thema Design-Prinzipien und erarbeiteten derer sieben. Diese Prinzipien konzentrieren sich auf das Design von KI-Tools für mehrere Outputs, die Berücksichtigung der möglichen Fehlerhaftigkeit von Outputs sowie den Aspekt der menschlichen Kontrolle, zum Beispiel in der Auswahl zwischen verschiedenen Output-Vorschlägen. Grundsätzlich sei die Erarbeitung von mentalen Modellen für generative KI-Anwendungen nötig und das Design gleichzeitig so auszurichten, dass gefährliche oder potenziell schadenstiftende Ausgaben verhindert werden. Dies überprüften Wang

et al. (2023) für ChatGPT, indem sie vom Modell Code für ein Phishing-Mail generiert haben wollten, das Tool eine derartige Ausgabe jedoch nicht unterstützte.

Sicherheit

Das Thema Sicherheit stellt in der Anwendung von generativer KI auch eine Herausforderung dar. In der Programmierung birgt das Verwenden von generiertem Code die Gefahr, dass dieser Sicherheitslücken und andere Schwachstellen aufweist (Chen et al., 2021). Um dem entgegenzuwirken, schlagen Chen et al. (2021) Reviews durch qualifizierte Anwender:innen vor. Je nach Anwendungsfall ist es zudem kritisch, wenn sensible Daten vom Modell verarbeitet werden, da diese in Ausgaben entsprechend vorhergesagt werden können. Chen et al. (2021) gehen im Falle von Codex und den öffentlich zugänglichen Repositorien, mit welchen Codex trainiert wurde, davon aus, dass alle sensiblen Daten in den Trainingsdaten bereits kompromittiert wurden. Weiter betonen Chen et al. (2021) wie vorab bereits erwähnt, dass generative KI-Tools zur Unterstützung von Cyberkriminalität verwendet werden kann. Die Relevanz dieses Themas nimmt mit zunehmender Leistungsfähigkeit der LLMs zu, da diese entsprechend auch besseren bzw. gefährlicheren Code liefern könnten. In diesem Kontext gehen Chen et al. (2021) auch davon aus, dass leistungsfähigere Modelle zur Codegenerierung auch für die Entwicklung von Malware förderlich sein könnten. Dies wird insbesondere dann gefährlich, wenn die Software-Vielfalt abnimmt, da mehr generierter Code mit ähnlichen Mustern, mit denen die KI trainiert wurde, im Umlauf ist. Deng & Lin (2022) pflichten dem bei und sehen das Risiko von Angriffen oder Manipulation am Modell zum Erzeugen von unerwünschtem Output als eines der Hauptrisiken. Zudem gehen Deng & Lin (2022) auf die Verbreitung von Fehlinformationen durch ChatGPTs Fähigkeit, menschenähnlichen Text zu generieren, ein. Für Unternehmen sei es daher entscheidend, Risiken sorgfältig abzuwägen und Massnahmen zu ergreifen, um diese beim Einsatz von KI-Tools zu minimieren.

Rechtliche Aspekte

Auch rechtliche Herausforderungen dürfen in Bezug auf generative KI nicht aussen vorgelassen werden. Eine potenzielle Herausforderung ist das Beschaffen und die Verwendung der Trainingsdaten, da diese Daten je nach Zugänglichkeit geschützt sind oder auch schützenswert sein könnten. Laut O'Keefe (2019, zitiert nach Chen et al., 2021) wurde das Training von KI-Modellen mit Internetdaten, beispielsweise öffentlichen GitHub-Repositorien, aber als ein Fall von "fair use", also einer erlaubten Verwendung, identifiziert. Nach Ziegler (2021, zitiert nach Chen et al., 2021) lag die Häufigkeit der Fälle, bei denen generierter Code mit Codeartefakten aus den Trainingsdaten

übereinstimmten, bei unter 0,1 Prozent. Das Risiko, identische Inhalte zu verwenden oder zu publizieren, ist daher als äusserst gering einzustufen.

Umwelt

Ein weiterer von Brown et al. (2020) und Chen et al. (2021) ins Spiel gebrachter Faktor, ist das Thema Ressourcen und Energieverbrauch von LLMs, der gerade zu Zeiten von Energiekrisen und Diskussionen zum Klimawandel an Relevanz gewinnt. Gross angelegte Pre-Trainings von Sprachmodellen erfordern viel Rechenleistung. Das Pre-Training des GPT-3 Modells mit 175 Milliarden Parametern hat mehrere tausend Petaflop/s-Tage an Rechenleistung verbraucht, im Vergleich zu nur wenigen Petaflop/s-Tagen für ein 1,5-Milliarden-Parameter GPT-2-Modell (Brown et al., 2020). Das lässt nur vermuten, wie viel Energie das Training des GPT-4 Modells mit deutlich mehr Parametern als GPT-3, verbraucht haben könnte. Laut Chen et al. (2021) hat auch das Fine-tuning von Codex eine ähnliche Menge an Energie verbraucht. Ihnen zufolge wurde das Training des Modells auf Microsofts Plattform Azure durchgeführt und der CO2-Fussabdruck durch den Erwerb von Umweltzertifikaten reduziert.

4.3 Zusammenfassung

In diesem Kapitel wurde beschrieben, was für eine Rolle generative KI in verschiedenen Anwendungsgebieten der Programmierung einnehmen kann. Diese Anwendungsgebiete lassen sich anhand der Überbegriffe Codegenerierung, Codeübersetzung, Fehlerbehebung und Codeverständlichkeit zusammenfassen. Grossmehrheitlich sind Forschende zum Schluss gekommen, dass KI-Tools wie GitHub Copilot oder ChatGPT innerhalb dieser Anwendungsgebiete vor allem eine unterstützende Rolle einnehmen, Programmierer:innen aber nicht gänzlich ersetzen können. Für die Beantwortung der Frage *„Welches sind die Herausforderungen bei der Nutzung von generativer KI zur Migration von Programmiersprachen?“* wurden generell Potenziale und Herausforderungen bezüglich generativer KI in der Programmierung betrachtet. Eine dieser Herausforderungen ist das Bilden von Vertrauen in KI-Tools. Dafür müssen geeignete User Interfaces geschaffen und gleichzeitig sichergestellt werden, dass die Leistungsfähigkeit des eingesetzten Tools den Programmierer:innen einen Mehrwert bringt. Im speziellen Anwendungsfall der Codemigration bzw. -übersetzung beschreiben Weisz et al. (2022), dass KI-Tools ein gutes Gerüst bilden und in einer unterstützenden Rolle durchaus Mehrwert generieren. Das Stichwort Vertrauen kann zwei Ausprägungen haben: Einerseits müssen Programmierer:innen Vertrauen in das Tool haben, um es überhaupt zu verwenden, andererseits ist blindes Vertrauen gefährlich und kann diverse negative Auswirkungen haben. Eine

weitere grosse Herausforderung ist das Thema Voreingenommenheit. KI-Modelle generieren ihren Output lediglich basierend auf den Daten, mit denen sie trainiert wurden und sind daher voreingenommen. Im Kontext der Programmierung beschreiben Liu et al. (2023) beispielsweise, dass ChatGPT Zugang zu deutlich mehr Trainingsdaten für verbreitete Programmiersprachen wie Python, C++ und Java als für Sprachen wie COBOL hatte. Entsprechend könnte ChatGPT für gewisse Programmiersprachen und -stile ungeeignet sein. In der Codeübersetzung könnten sich derartige Stil-Probleme zeigen, da COBOL und Python in ihren Grundzügen anderen Konzepten unterliegen: Während COBOL grundsätzlich eine prozedurale Programmiersprache ist und Konzepte der Objektorientierung erst später eingeführt wurden, ist Python eine objektorientierte Programmiersprache, die aber auch andere Programmierstile unterstützt. Weitere Herausforderungen mit generativer KI entstehen durch Probleme mit dem Erkennen von Zusammenhängen und logischem „Denken“, was im Kontext von Codeübersetzungen durchaus problematisch sein kann. Es ist dadurch möglich, dass das Modell die eigentlich angedachte Funktionalität bzw. die Absicht von Programmierer:innen nicht erkennt und dadurch Übersetzungsfehler verursacht. In diesem Bereich ist ein gezieltes Prompt Engineering hilfreich, da so zusätzliche Informationen und der Kontext für ein Codeartefakt mitgegeben werden können. Herausforderungen in Richtung Sicherheit, rechtlicher Rahmen und Umwelt kommen hinzu. Diese Faktoren sind für die Untersuchungen im Rahmen dieser Arbeit aber nicht im Fokus. Die Betrachtung des Themas Sicherheit würde erst bei grösseren Projekten und nicht in sich geschlossenen Codeartefakten relevant werden. Laut Chen et al. (2021) können einige dieser Herausforderungen und Risiken durch eine sorgfältige Dokumentation und Gestaltung der Benutzeroberfläche, Anforderungen an Codereviews und Inhaltskontrollen (Filterung von Output) reduziert werden.

Die Aktualität des Themas generative KI, die vielfältigen Einsatzgebiete und das enorme wirtschaftliche Potenzial lassen darauf schliessen, das Unternehmen bereit sind, die beschriebenen Herausforderungen anzugehen und zu überwinden, um die positiven Aspekte dieser eher neuen Technologie zu nutzen.

5 Code- und Übersetzungsqualität

Das folgende Kapitel behandelt die zweite Forschungsunterfrage *„Anhand welcher Methoden und Qualitätskriterien werden teilautomatisierte Code-Migrationen durchgeführt und bewertet?“* Dafür werden die Themen Code- und Übersetzungsqualität betrachtet. Der Begriff Codequalität ist anhand der sechs Software-Qualitätseigenschaften Funktionalität, Zuverlässigkeit, Benutzbarkeit und Gebrauchstauglichkeit, Effizienz, Wartbarkeit und Übertragbarkeit in der ISO-9126-Norm beschrieben (Johner, 2015). Im wissenschaftlichen Kontext wurden die Themen Codequalität, Code-Qualitätsanalyse oder die Klassifikation von Codequalität anhand von Software Metriken schon weitläufig untersucht. Ein weiteres Untersuchungsgebiet stellt die Verwendung von (generativen) KI-Modellen im Bereich der Programmierung und die Evaluierung des gelieferten Outputs dar. Ein darin enthaltener Anwendungsfall ist die Übersetzung von Code von einer in eine andere Programmiersprache.

Im Folgenden wurde zunächst eine Literaturrecherche durchgeführt, um Kriterien und Metriken zur Messung von Software- und Codequalität zu definieren. Basierend auf der Recherche wurden diese in Kapitel 5.1 in sechs Überbegriffe gruppiert, beschrieben und letztlich auf den Anwendungsfall der Codeübersetzung fokussiert. Diese Kriterien wurden in Kapitel 5.2 mit ergänzenden Kriterien, welche dem vergleichenden Aspekt der Codemigration bzw. -übersetzung gerecht werden, erweitert. In Folge wird ausserdem auf Limitationen und mögliche Erweiterungen des Bewertungsmodells und -vorgehens eingegangen.

5.1 Kriterien und Metriken zur Bewertung von Codequalität

Code, der Best Practices folgt, kann Vorteile in Bereichen wie Wiederverwendbarkeit, Verständlichkeit oder Sicherheit geben. Deshalb wird das Thema in der Wissenschaft aus diversen Blickpunkten behandelt. Dalla Palma et al. (2020) haben im Rahmen ihrer Forschung einen Katalog aus Qualitätsmetriken für Infrastruktur-Code aufgestellt. Klima et al. (2022) befassten sich mit Codequalitätskriterien für IoT Applikationen und Rosenberg & Hyatt (1997) fokussierten insbesondere auf relevante Metriken für objektorientierte Programmierumgebungen. Baggen et al. (2012) und Rentrop (2006) untersuchten Software-Metriken als Benchmarks für die Qualität von Quellcode und Buse & Weimer (2010), AlOmar et al. (2019) und Meirelles et al. (2010) konzentrierten sich in erster Linie auf Verbesserungen in den Bereichen Leserlichkeit, Design und Attraktivität des Codes, welche durch eine höhere Codequalität erreicht werden kann. Dabei wird sich oft an den Qualitätskriterien aus der ISO-9126-Norm orientiert. So beschreiben Kanellopoulos et al.

(2010) eine Methodik zur Bewertung von Codequalität basierend auf der Norm und Stamelos et al. (2002) orientieren sich bei der Untersuchung von Open Source Entwicklung ebenfalls an ISO-9126.

In Folge wird genauer auf sechs Qualitätskriterien eingegangen, die sich basierend auf der Literaturrecherche als für den Anwendungsfall relevant erwiesen haben. Für jedes dieser Kriterien werden Unterkriterien und Metriken zur nachvollziehbareren Bewertung herangezogen. Metriken sind dabei als quantifizierbare Messgrössen zu verstehen und werden im Folgenden jeweils da verwendet, wo eine Quantifizierung sinnvoll ist. Andere Metriken die oft genannt, für den Anwendungsfall aber nicht relevant sind, werden nicht näher thematisiert.

5.1.1 Funktionalität

Das Kriterium Funktionalität beinhaltet die Sub-Charakteristiken Angemessenheit, Richtigkeit, Interoperabilität, Sicherheit und Konformität (Johner, 2015). Für die Bewertung von einfachen Codebeispielen im Kontext von Übersetzungen eignet sich das Kriterium Richtigkeit (accuracy). Diese wird bewertet, indem geprüft wird, ob der Code in der Zielsprache die gleichen Aufgaben erfüllt wie in der Ausgangssprache (Mittal & Bhatia, 2013). Nguyen & Nadi (2022) und Lachaux et al. (2020) zufolge ist es im Sinne der Richtigkeit auch wichtig zu überprüfen, ob In- und Outputs der Funktionen bzw. des Codes übereinstimmen.

5.1.2 Grösse

Im Kontext von Codeübersetzungen kann das Kriterium Grösse so angewendet werden, dass die Grösse des Ausgangscodes mit jener der Ausgabe verglichen wird, wobei eine Optimierung im Sinne von kompakterem Code wünschenswert wäre. Auf den Anwendungsfall sinnvolle und anwendbare Metriken sind “Lines of Code (LOC)“ und “Number of blank lines” (AlOmar et al., 2019; Breuker et al., 2011; Dalla Palma et al., 2020; Kanellopoulos et al., 2010; Klima et al., 2022; Meirelles et al., 2010; Rentrop, 2006; Rosenberg & Hyatt, 1997; Sharma & Spinellis, 2020; Stamelos et al., 2002). Lines of Code beschreibt die totale Anzahl mit ausführbarem Code und Number of blank lines zählt die leeren Zeilen im Code. Zur Bewertung der Grösse der Übersetzung werden diese Metriken der Eingabe mit jener der Ausgabe von ChatGPT verglichen. Eine weitere denkbare Metrik wäre das „Halstead Volume“ gewesen. Einer Umfrage von Sharma & Spinellis (2020) zufolge wird dieses in der Praxis jedoch kaum verwendet und daher nicht mit einbezogen.

Grundsätzlich misst das Halstead Volume die Grösse eines Programmes basierend auf der Anzahl an verwendeten Operatoren und Operanden im Code.

5.1.3 Redundanz

Redundanter Code gilt aus Gründen der Leserlichkeit und Übersichtlichkeit als schlechter Code. Eine Codezeile gilt dann als redundant, wenn sie Teil eines Codefragments von mindestens sechs Codezeilen ist, das sich wörtlich an einer anderen Stelle im Code wiederholt (Baggen et al., 2012). In diesem Bereich wird die „Code Duplication (CD)“ und das „Percentage of redundant code“ gemessen. Code Duplication ist als absolute Anzahl an sich wiederholendem Code zu verstehen. Die prozentuale Anzahl an sich wiederholendem Code berechnet sich aus dem Wert Code Duplication, welcher durch die Lines of Code geteilt wird (Baggen et al., 2012; Breuker et al., 2011; Klima et al., 2022; Rentrop, 2006). Baggen et al. (2012) ziehen die „Percentage of redundant code“ als Messgrösse heran und bewerten anhand eines Fünfsterne-Systems. Code der maximal 3 Prozent Redundanz aufweist, wird dabei als ideal bewertet. Vier Sterne wird bei Duplikation bis 5 Prozent gegeben. Drei respektive zwei Sterne werden bei den Schwellwerten 10 und 20 Prozent erreicht. Enthält der Code mehr als 20 Prozent redundante Stellen, erhält er das schlechteste Rating (Baggen et al., 2012).

5.1.4 Komplexität

Komplexität ist ein wichtiges Mass bei der Bewertung von Codequalität, da gerade eine hohe Komplexität negative Auswirkungen auf diverse andere Eigenschaften wie Verständlichkeit, Wartbarkeit oder Nutzbarkeit haben kann. Das prominenteste Mass für die Berechnung der Codekomplexität ist die „Cyclomatic Complexity“. Dabei wird die Komplexität eines Codes anhand der Anzahl von Verzweigungen beurteilt. Eine Verzweigung kann dabei beispielsweise eine if-Bedingung sein. Ein hoher Wert deutet auf eine hohe Komplexität des Codes hin. Weiter kann „Number of methods / functions“ herangezogen werden. Je mehr Funktionalitäten von einem Codeausschnitt abgebildet werden, desto höher wird auch die Komplexität gewertet (AlOmar et al., 2019; Baggen et al., 2012; Klima et al., 2022; Meirelles et al., 2010; Rentrop, 2006; Sharma & Spinellis, 2020; Stamelos et al., 2002; Vytovtov & Markov, 2017). Baggen et al. (2012) assoziieren eine hohe Komplexität mit einem höheren Risiko. Werte zwischen 1 bis 10 stufen sie als geringes Risiko, Werte von 11 bis 20 als moderates Risiko, von 21-50 als hohes und höhere Werte als 50 als sehr hohes Risiko ein.

5.1.5 Verständlichkeit

Wie im vorherigen Abschnitt erwähnt, können diverse andere Eigenschaften einen Einfluss auf die Verständlichkeit von Code haben. Gerade für Dritte kann die Verständlichkeit aber durch Kommentare im Code erhöht werden. Deshalb eignen sich hierfür die Metriken „Number of comments“, „Percentage of comment lines“ und „comment quality“. Number of comments misst die Anzahl an Kommentaren im Code. Die Percentage of comment lines misst das Verhältnis von Kommentaren zu Zeilen an ausführbarem Code. Als effektivsten Wert beschreiben Rosenberg & Hyatt (1997) einen Kommentaranteil von rund 30 Prozent. Die Qualität von Kommentaren ist qualitativ zu bewerten und entsprechend zu prüfen, ob ein Kommentar zur besseren Verständlichkeit des Codes beiträgt. Je nach betrachtender Person hat diese Art der Beurteilung eine subjektive Komponente (Dalla Palma et al., 2020; Klima et al., 2022; Meirelles et al., 2010; Rosenberg & Hyatt, 1997; Sharma & Spinellis, 2020; Stamelos et al., 2002). Ein weiteres Kriterium, dass zur Verständlichkeit von Code beitragen kann, ist die Lesbarkeit. Klima et al. (2022) beschreiben die Lesbarkeit als Leichtigkeit, den Code zu verstehen. Grundsätzlich ist dieses Kriterium subjektiver Natur und hängt von der Erfahrung der Programmierer:innen ab. Buse & Weimer (2010) geben beispielsweise die Zeilenlänge, dass saubere Einrücken aber auch das Verwenden von Kommentaren als Metriken zur Bewertung des Kriteriums Lesbarkeit an. Ebenfalls relevant für die Lesbarkeit sind die Anzahl an Leerzeilen (Breuker et al., 2011) und die Namensgebung von Variablen und Funktionen, welche die Lesbarkeit negativ beeinflussen können, wenn sie zu lange sind oder keine zusätzlichen Informationen vermitteln (Buse & Weimer, 2010).

5.1.6 Wartbarkeit und Zuverlässigkeit

Die Wartbarkeit und Zuverlässigkeit sind weitere Kriterien, die gerne für die Bewertung von Codequalität verwendet werden, wobei sie verschiedene Ausprägungen haben können. Basierend auf Johner (2015) gehören die Unterkriterien Stabilität, Änderbarkeit, Testbarkeit und Analysierbarkeit zur Wartbarkeit und Laufzeit, Wiederherstellbarkeit und Fehlertoleranz gehören zur Zuverlässigkeit. Wie bei den anderen Kriterien können diverse Faktoren Einfluss auf die Wartbarkeit haben. So wird diese beispielsweise von der Grösse des Programms beeinflusst – je länger der Code, desto schwieriger die Wartung (Rosenberg & Hyatt, 1997). Meirelles et al. (2010) nennen zusätzlich die Flexibilität als Faktor für die Wartbarkeit. Ihnen zufolge kann die Wartbarkeit eines Programms mit Metriken wie der Anzahl an verwendeten Funktionen oder der Anzahl Variablen gemessen werden. Die Zuverlässigkeit, insbesondere die Unterkriterien Wiederherstellbarkeit und

Fehlertoleranz gehen mit dem vorab genannten Begriff „program repair“ einher. Hiernach kann bewertet werden, ob sich fehlerhafter Code effizient reparieren und in einen funktionsfähigen Zustand bringen lässt

5.2 Bewertung von KI-unterstützten Codeübersetzungen

Die vorab definierten Kriterien und Metriken können zur Bewertung von kurzen, in sich geschlossenen Codebeispielen verwendet werden. Im Bereich der Codeübersetzungen ist jedoch ein angepasstes Vorgehen nötig. Nimmt man beispielsweise die Cyclomatic Complexity als Mass und zählt, wie viele if-Bedingungen ein Code enthält, dann ist bei einer Übersetzung von Code davon auszugehen, dass im ursprünglichen und im übersetzten Code die gleichen Funktionalitäten abgebildet sind und entsprechend gleich viele Verzweigungen im Code enthalten sind. Dennoch können einige Metriken als quantifizierbare Messgrössen dienen und der Subjektivität der qualitativen Auswertung von Codequalität entgegenwirken. In diesem Abschnitt werden Aspekte der automatisierten Codegenerierung und -übersetzung hinzugezogen, um so letztlich passende Kriterien für den gewählten Anwendungsfall zu definieren.

Moradi Dakhel et al. (2022) haben untersucht, ob GitHub Copilot als unterstützendes KI-Tool Programmierer:innen einen Mehrwert bringt oder nicht. Dies wurde anhand von vier Eigenschaften bewertet: Erfolgreiche Codegenerierung, Richtigkeit des Codes, Optimalität des Codes und dessen Reproduzierbarkeit. Diese Eigenschaften können für die Bewertung von Übersetzungen adaptiert werden, wobei Moradi Dakhel et al. (2022) für ihre Untersuchungen Metriken verwendet haben, die eher für neu generierten Code und eine grössere Stichprobe Sinn machen. Zu den verwendeten Metriken gehören die Correct Ratio (CR), Repair Rate, Average Repair Time, Relative Patch Size (RPS), Cyclomatic Complexity und die „pass@topk“ Metrik. Die „pass@k“ Metrik und deren angepassten Formen wie pass@topk werden auch von Chen et al., (2021) verwendet, wobei für eine aussagekräftige Verwendung eine grosse Stichprobe verwendet werden muss. Die pass@k Metrik bemisst dann, wie viele korrekte Samples im Verhältnis zur gesamten Stichprobe generiert wurden, auch indem für das Lösen desselben Problems eine Vielzahl von Codebeispielen herangenommen wird. Pudari (2022) verfolgt hingegen einen nützlichen Ansatz und unterteilt Software bzw. Code in ihrer Arbeit in verschiedene Abstraktionsebenen und untersucht, in welchen Ebenen KI-Modelle Aufgaben gut lösen können. Pudari (2022) nennt Syntax als tiefste Abstraktionsebene, auf welcher es lediglich darum geht, einen fehlerfreien, kompilierbaren Code zu erhalten. Auf zweiter Ebene wird erneut die Richtigkeit genannt, wobei es noch nicht darum geht, optimalen Code zu

produzieren, sondern gestellte Probleme zu lösen. Das dritte Level sind Paradigmen und Idiome, gefolgt von Code Smells, Module Level Design und System Level Design. Gemäss Pudari (2022) erfüllen KI-Modelle die ersten beiden Abstraktionsebenen mit gutem Erfolg. Sobald es darum geht, optimalen Code zu generieren oder auf dem Level Code Smells schlechte Praktiken zu vermeiden, stossen KI-Modelle an ihre Grenze. Hierbei wird allerdings jeweils die erste Ausgabe des KI-Modells bewertet. Szafraniec et al. (2023) haben die Codeübersetzung mit Compilern untersucht und sind dabei zum Ergebnis gekommen, dass mit den richtigen Anweisungen und der richtigen Methodik die Anzahl an korrekten Ausgaben um 11 % erhöht werden kann. Es wird zwischen einem „unsupervised“ und einem „supervised“ Ansatz unterschieden. Weitere Erkenntnisse von Vaithilingam et al. (2022) und Moradi Dakhel et al. (2022) zeigen, dass KI-Modelle nicht immer optimale Ausgaben generieren, diese aber durchaus als guter Startpunkt für die Weiterverwendung verwendet werden können oder mit minimalen manuellen Korrekturen zum richtigen Ergebnis führen können.

Um Codeübersetzungen bewerten zu können, ist es wichtig zu verstehen, wo potenzielle Fehlerquellen im Code liegen können. Weisz et al. (2022) haben die KI-unterstützte Codeübersetzung in einem anderen Kontext bereits untersucht und dabei eine Taxonomie für potenzielle Fehler in Übersetzungen aufgestellt. Es wird dabei zwischen Translation Error (TE), Language Error (LE), Spurious Error (SE), Code Omission Error (COE), Documentation Omission Error (DOE) und Correctness Error (CE) unterschieden (vgl. Abbildung 6).

Error	Description	Operationalization	Source Error(s)
Translation Error (TE)	Participant mistranslated a code statement by making an error in an assignment statement, a conditional statement, a looping conditional, an array lookup, whitespace, or other logical statement	Count the number of code segments that needed to be modified to fix assignments, conditionals, loops, array lookups, etc.	Translation error, logic error [70]; Assignment bug, Iteration bug, Array bug [36]; Logical bug [28]; Lexical bugs [29]
Language Error (LE)	Participant included snippets of Java code within Python or failed to appropriately translate Java language idioms to Pythonic idioms	Count the number of code segments that needed to be modified because Java idioms were used or Python requirements were not met	Dummy bug [28]; Language liability [49]; Language [29]
Spurious Error (SE)	Participant included functionality not part of the original Java program (e.g. by defining new methods)	Count the number of irrelevant, unnecessary, or extraneous code statements	Spurious [44]
Code Omission Error (COE)	Participant omitted the translation of a method or code statements within a method, or provided a trivial implementation (e.g. `pass`, `return None`, `print(''not implemented'')`, etc.)	Count the number of instances in which code was added due to missing, trivial, or incomplete method implementations	Missing [44]; Forgotten function [49]; Omission error [70]
Documentation Omission Error (DOE)	Participant omitted translation of a function's documentation (e.g. Javadoc comment)	Count the number of Python classes and methods that were missing documentation present in the Java source	Missing [44]; Omission error [70]
Correctness Error (CE)	Participant's translation of a method was incorrect (e.g. did not pass unit tests)	Count the number of methods that required one or more modifications to pass unit tests, including methods that weren't implemented	Algorithm awry [49]

Abbildung 6: Beschreibung von Fehlerquellen für Codeübersetzungen. (Weisz et al., 2022, S. 375)

Auch wenn die Untersuchungen von Weisz et al. (2022) auf Übersetzungen zwischen Java und Python fokussieren, können diese Überlegungen auf andere Sprachpaare adaptiert werden. Die in Abbildung 6 gelisteten Fehlerquellen gilt es bei der Auswertung der durchgeführten Codeübersetzungen zu berücksichtigen.

5.3 Methodik und Modell

Nun wird basierend auf der durchgeführten Literaturrecherche in Anlehnung an das Vorgehen von Nguyen & Nadi (2022), Geng et al. (2023), Moradi Dakhel et al. (2022) und Weisz et al. (2022) der Aufbau der empirischen Evaluation abgeleitet.

Nguyen & Nadi (2022) evaluierten im Rahmen ihrer Untersuchung Codevorschläge von GitHub Copilot. Dies haben sie anhand von vier Schritten durchgeführt. Zunächst werden die für die empirische Studie nötigen Voraussetzungen geschaffen und nötige Queries generiert. Dann werden Codevorschläge mit GitHub Copilot generiert, welche anschliessend evaluiert und bewertet werden. Geng et al. (2023), die versucht haben mit ChatGPT Aufgaben aus einem Einführungskurs in die Programmierung zu absolvieren, erweitern das Vorgehen um eine Dimension, in dem sie ChatGPT „unassisted“ und „assisted“ verwenden. Das von Moradi Dakhel et al. (2022) genannte Kriterium der Reproduzierbarkeit lässt sich anhand dieser zwei Dimensionen testen, weil dabei überprüft werden kann, ob voneinander unabhängige Durchläufe, in diesem Fall Übersetzungen, zu einem Ergebnis mit der gleichen Struktur und Funktionalität führen. Weisz et al. (2022) geben zudem Anhaltspunkte für mögliche zu beachtende Fehlerquellen, welche in der Bewertung relevant sein können.

5.3.1 Vorgehen nach Phasen

Im Rahmen dieser Arbeit wird in einem empirischen Setup ChatGPT-4 als unterstützendes Tool zur Übersetzung von Code zwischen zwei Programmiersprachen – COBOL und Python - untersucht und bewertet. In Folge ist die abgeleitete Methodik anhand von drei Phasen beschrieben.

Sammeln von COBOL Codebeispielen

Im ersten Schritt wird eine Codebasis mit COBOL Codebeispielen erstellt. Für die systematische Datensammlung wurden zunächst Auswahlkriterien für die Codebeispiele festgelegt. Die ausgewählten Codebeispiele sollen grundlegende Funktionalitäten einer Programmiersprache, wie Schleifen, Bedingungen, Datenmanipulation, Ein- und Ausgaben und Berechnungen abdecken. Ausserdem müssen die Codebeispiele in sich

geschlossen und verständlich, sowie in einem Online-Kompilierer kompilierbar sein. Anschliessend wurde basierend auf diesen Kriterien eine Recherche auf GitHub durchgeführt, um öffentlich zugängliche Repositorien zu identifizieren, die potenziell geeignete COBOL-Codebeispiele enthalten. In Folge wurde jedes ausgewählte Codebeispiel überprüft, um festzustellen, ob dieses den Auswahlkriterien entspricht. Die Überprüfung wurde mit den Online-COBOL-Kompilierern von OneCompiler (*https://onecompiler.com/cobol*) und tutorialspoint (*https://www.tutorialspoint.com/compile_cobol_online.php*) durchgeführt. Anschliessend wurden die Codebeispiele im GitHub Repository *https://github.com/stuufc/codesamples.git* gesammelt. Dort wurden die Codebeispiele mit einer kurzen Beschreibung erweitert. Ausserdem sind Quellen der Codebeispiele darin zu entnehmen.

Übersetzung des COBOL Code mit ChatGPT

Im zweiten Schritt werden die gesammelten COBOL Codebeispiele mit GPT-4 in die Zielsprache Python übersetzt. Hierbei wird der Ansatz von Geng et al. (2023) verfolgt und zwei Durchläufe gemacht. Im ersten Durchlauf werden die Codebeispiele ohne Anweisungen zur Übersetzung in das Tool eingegeben. Im zweiten Durchlauf wird der Prompt mit Anweisungen in natürlicher Sprache angereichert, so dass die Ausgabe beispielsweise mit Kommentaren versehen oder auf Best Practices geachtet werden soll.

Prompt ohne Anweisung	**Prompt mit Anweisung**
Übersetze den folgenden COBOL Code in Python Code	Übersetze den folgenden COBOL Code in Python Code. Achte dabei insbesondere auf die Beibehaltung der Funktionalität und darauf, dass der Code möglichst optimal ist, Best Practices einhält und möglichst verständlich ist. Füge wo nötig Kommentare hinzu.

Tabelle 2: Verwendetes Prompt Engineering für die Evaluation von ChatGPT.

Bewertung des generierten Python Codes

Der generierte Python Code wird anhand der vorab abgeleiteten Kriterien (vgl. Abbildung 8, 9 & 10) analysiert und entsprechend die Qualität der Übersetzungen evaluiert. Ziel dabei ist die qualitative Bewertung von ChatGPT als Übersetzungstool, um die Eignung und Fähigkeiten in diesem Kontext einordnen zu können. Der generierte Python Code wird dafür in Visual Studio Code kompiliert und mit dem ursprünglichen COBOL Code verglichen. Die Evaluation wird basierend auf dem in Kapitel 5.3.2 eingeführten Modell zur Bewertung durchgeführt. Der Code, den ChatGPT generiert hat, wird dafür wo möglich mit den gleichen Testfällen (z. B. identischer Input / Output) wie das COBOL

Programm getestet und verglichen. Jedes Bewertungsraster enthält zudem ein Kommentarfeld, worin Auffälligkeiten aus dem Evaluationsprozess festgehalten werden.

5.3.2 Modell zur Bewertung

Für die qualitative Bewertung von ChatGPT-4 als Codeübersetzungstool wurden die vorab erarbeiteten Kriterien, Unterkriterien und Metriken in drei für die Evaluation relevante Hauptkriterien zusammengefasst. Abbildung 7 zeigt die Herleitung der Kriterien. Dabei fällt insbesondere auf, dass sich diverse Kriterien gegenseitig beeinflussen. Die Beziehungen zwischen den Kriterien wurde mit Pfeilen dargestellt, wobei die gestrichelten Linien indirekte Auswirkungen auf das zusammengefasste Hauptkriterium signalisieren.

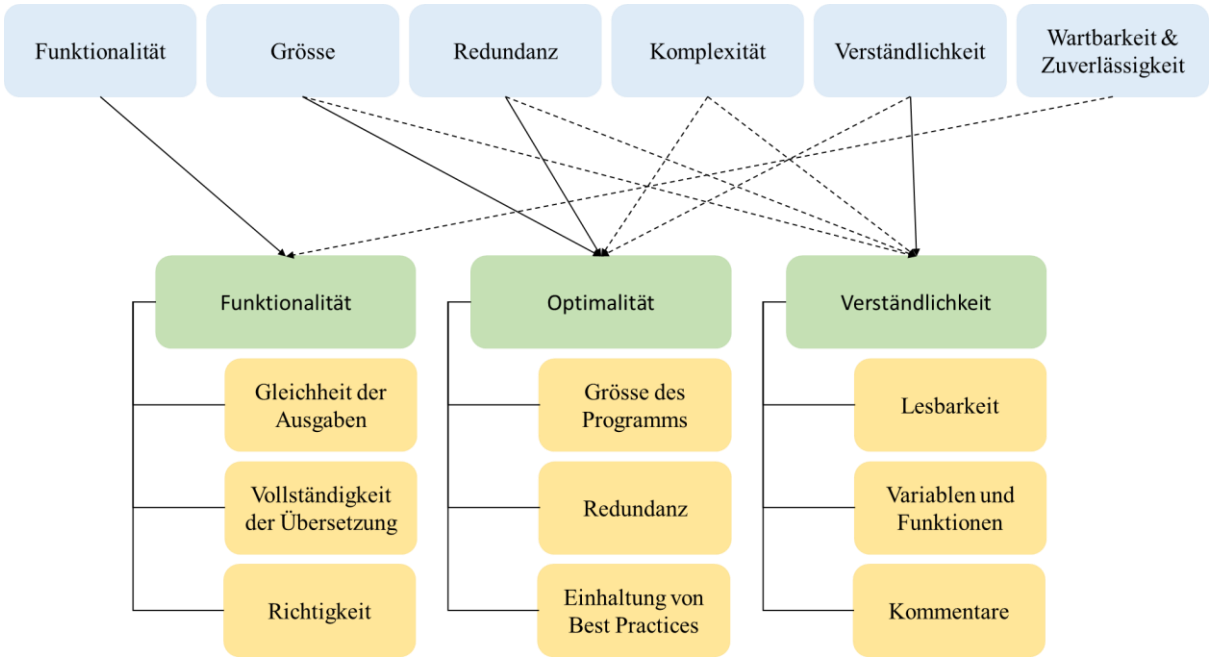


Abbildung 7: Herleitung der Bewertungskriterien zur Evaluation von ChatGPT-4. (eigene Darstellung)

Für eine bessere Nachvollziehbarkeit und ein strukturierteres Vorgehen bei der Bewertung der Übersetzungen wurden pro Hauptkriterium Unterkriterien abgeleitet, für die zwischen 0 bis 2 Punkte erreicht werden können – also maximal 6 Punkte pro Hauptkriterium. Die Einführung dieser einfachen Scoring-Logik wird verwendet, um die Ergebnisse vergleichbar zu machen.

Vergebene Punkte	**Bedeutung**
2	Kriterium vollständig erfüllt
1	Kriterium teilweise erfüllt
0	Kriterium nicht erfüllt

Tabelle 3: Punktevergabe nach Erfüllung von Kriterien im Bewertungsraster

Das Hauptkriterium Funktionalität enthält die drei Unterkriterien „Gleichheit der Ausgaben“, „Vollständigkeit der Übersetzung“ und „Richtigkeit“. Ausprägungen der jeweiligen Unterkriterien sind Abbildung 8 zu entnehmen. Dieses Kriterium zielt darauf ab, herauszufinden, ob die durchgeführten Übersetzungen akkurat sind oder nicht. Deshalb ist es wichtig, zu prüfen, ob der Code bei gleichen Inputs die gleichen Ausgaben produziert, ob die Übersetzung vollständig durchgeführt wurde und ob der Code richtig übersetzt wurde. Das Unterkriterium Richtigkeit berücksichtigt bei fehlerhaftem Verhalten auch die Fähigkeit von ChatGPT, Fehler im Code zu beheben, sollten diese auftreten.

Punkte	Funktionalität		
	Gleichheit der Ausgaben	**Vollständigkeit der Übersetzung**	**Richtigkeit**
2	Die Ausgaben des Python-Codes und des COBOL-Codes sind identisch.	Der Python-Code übersetzt und implementiert alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert ohne Fehler und alle Funktionen können ohne Ausführungsfehler ausgeführt werden.
1	Die Ausgaben des Python-Codes und des COBOL-Codes sind ähnlich, aber nicht identisch.	Der Python-Code übersetzt und implementiert die meisten, aber nicht alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert mit einigen Warnungen, oder es treten bei der Ausführung einige kleinere Fehler auf, die jedoch die grundsätzliche Funktionsfähigkeit nicht beeinträchtigen. Fehler lassen sich mit wenigen zusätzlichen Prompts beheben.
0	Die Ausgaben des Python-Codes und des COBOL-Codes unterscheiden sich erheblich.	Der Python-Code lässt wesentliche Funktionen, Verfahren oder Operationen des ursprünglichen COBOL-Codes aus oder implementiert sie falsch.	Der Python-Code kompiliert nicht, oder es treten bei der Ausführung schwere Fehler auf, die die Funktionsfähigkeit beeinträchtigen. Fehler lassen sich kaum beheben

Abbildung 8: Bewertung des Kriteriums Funktionalität anhand von Unterkriterien. (eigene Darstellung)

Das Hauptkriterium Optimalität enthält die drei Unterkategorien „Grösse des Programms“, „Redundanz“ und „Einhaltung von Best Practices“. Ausprägungen der jeweiligen Unterkriterien sind Abbildung 9 zu entnehmen. Die Bezeichnung Optimalität wird dabei als Überbegriff verwendet und kann mitunter eine subjektive Bewertung mit sich ziehen. Ziel ist es, herauszufinden, ob der übersetzte Code optimal und dementsprechend verwendet werden würde, oder ob dieser von Programmierer:innen angepasst werden müsste. Im Kontext dieser Erhebung kann dies evaluiert werden, indem betrachtet wird, ob die Ausgabe mehr oder weniger Zeilen (und Leerzeilen) beinhaltet, ob redundanter Code geschrieben wurde und zuletzt die Bewertung, ob der Code unter Einhaltung von verbreiteten Best Practices geschrieben wurde. Wenn die Grösse des Programms zwar kleiner ist als jene des Ausgangscodes, wird zusätzlich berücksichtigt, ob Code effizient und komprimiert geschrieben wurde, oder ob der Code in der Zielsprache zu ineffizient ist. Die Bewertung der Redundanz berücksichtigt auch, ob mit effektiveren Methoden ähnliche Codezeilen vermieden werden könnten.

	Optimalität		
	Grösse des Programms	**Redundanz**	**Einhaltung von Best Practices**
2	Der Python-Code ist effizient geschrieben und so gut wie möglich komprimiert. Es gibt eine angemessene Anzahl von Leerzeilen zur Strukturierung.	Der Python-Code vermeidet redundante Operationen oder Wiederholungen. Code-Wiederverwendung wird maximiert.	Der Python-Code hält sich an allgemein anerkannte Best Practices für Python-Programmierung, einschliesslich Einrückung, Benennungskonventionen, Verwendung von Funktionen und Klassen usw.
1	Der Python-Code ist im Allgemeinen effizient geschrieben, es gibt jedoch Möglichkeiten zur Optimierung.	Der Python-Code enthält einige redundante Operationen oder Wiederholungen, die reduziert werden könnten.	Der Python-Code hält sich grösstenteils an Best Practices, es gibt jedoch einige Verstösse.
0	Der Python-Code ist ineffizient und zu lang. Es gibt zu viele Leerzeilen.	Der Python-Code ist voller redundanter Operationen oder Wiederholungen.	Der Python-Code verstösst häufig gegen Best Practices.

Abbildung 9: Bewertung des Kriteriums Optimalität anhand von Unterkriterien. (eigene Darstellung)

Das Hauptkriterium Verständlichkeit enthält die drei Unterkategorien „Lesbarkeit“, „Variablen und Funktionen“ und „Kommentare“. Ausprägungen der jeweiligen Unterkriterien sind Abbildung 10 zu entnehmen. Die Verständlichkeit ist ein subjektives Kriterium und hängt von der Erfahrung der Programmierer:innen ab. Jedoch gibt es Praktiken, die zur verbesserten Verständlichkeit von Code beitragen. Eine gute Lesbarkeit und die Einhaltung von Namensgebungskonventionen für Variablen und Funktionen tragen positiv zur Verständlichkeit von Code bei, wohingegen unklare Bezeichnungen negativen Einfluss haben. Das gleiche gilt für den Inhalt von Kommentaren im Code. Zusätzlich kann hierfür für die Bewertung aber auch die Frequenz von Kommentaren herangezogen werden.

	Verständlichkeit		
	Lesbarkeit	**Variablen und Funktionen**	**Kommentare**
2	Der Python-Code ist einfach zu lesen und zu verstehen.	Variablen und Funktionen haben klare, beschreibende Namen, die ihrem Zweck im Code entsprechen.	Der Code ist gut kommentiert, und die Kommentare tragen effektiv zum Verständnis des Codes bei.
1	Der Python-Code ist verständlich, enthält aber unklare Abschnitte.	Die meisten Variablen und Funktionen haben klare, beschreibende Namen, aber einige könnten verbessert werden.	Der Code hat einige nützliche Kommentare, aber es gibt Bereiche, die mehr Kommentierung erfordern oder wo die vorhandenen Kommentare nicht klar sind.
0	Der Python-Code ist schwer zu lesen und zu verstehen.	Variablen und Funktionen haben unklare oder nicht beschreibende Namen.	Der Code hat wenige oder keine hilfreichen Kommentare.

Abbildung 10: Bewertung des Kriteriums Verständlichkeit anhand von Unterkriterien. (eigene Darstellung)

5.4 Einschränkungen und Einordnung

Die in dieser Arbeit erarbeitete Methodik zur Evaluation weist diverse Einschränkungen auf. Das Modell zur Bewertung wurde darauf abgestimmt, dass es sich für die ausgewählten Codebeispiele eignet. Die Codebeispiele wurden so ausgewählt, dass sie Grundfunktionalitäten einer Programmiersprache abbilden, in sich geschlossen und online kompilierbar sind. Für die Bewertung von Codequalität und Übersetzungsqualität in der Praxis müssten weitere Bewertungskriterien, die für diese Erhebung bewusst weggelassen wurden, hinzugezogen werden. Beispiele dafür sind Sicherheit, Effizienz und Portabilität. Die Aussagekraft dieser Evaluation für die Praxis, beispielsweise für Migrationen eines Kernbankensystems, ist aus diesen Gründen eingeschränkt. Diese Kriterien sind erst im komplexeren Kontext aussagekräftig. Das Modell wurde ausserdem so aufgestellt, dass es eine einfache Bewertungslogik enthält. Diese Logik dient in erster Linie

dazu, die Ausgabequalität von ChatGPT-4 im „unassisted“ und „assisted“-Ansatz zu vergleichen und zu bewerten. Des Weiteren werden Ausgaben von LLMs als inkonsistent beschrieben (Tian et al., 2023), was bedeutet, dass LLMs mit einer gewissen Zufälligkeit für die gleichen Fragen unterschiedliche Antworten ausgeben können. Das Ergebnis der Evaluation muss daher als Momentaufnahme betrachtet werden. Dieser Zufälligkeit wird mit dem Durchführen von zwei Übersetzungen entgegengewirkt. Weiter wurde für die Evaluation das ChatGPT-4 Modell der Release-Version vom 24. Mai 2023 verwendet. Ergebnisse könnten in weiteren Releases abweichen und mit grosser Wahrscheinlichkeit tendenziell besser ausfallen. Zuletzt wird die Fähigkeit von ChatGPT als Tool zur teilautomatisierten Codeübersetzung anhand eines ausgewählten Programmiersprachen-Paar getestet. Wie in Kapitel 4.2.2 beschrieben, wurde ChatGPT basierend auf den Trainingsdaten für gewisse Programmiersprachen besser trainiert als für andere. Es ist also davon auszugehen, dass Übersetzungen zwischen weit verbreiteten Programmiersprachen wie Java, C++ und Python, die zudem ähnlichen Grundlogiken und -ideen unterliegen, besser funktionieren, als wenn wenig verbreitete Programmiersprachen, die seltener in den Trainingsdaten vorkamen, verwendet werden.

6 Ergebnisse

Die Evaluation von ChatGPT-4 als Tool zur teilautomatisierten Codeübersetzung wurde durchgeführt, um folgende Forschungsunterfrage zu beantworten: *„Inwiefern erfüllen die von ChatGPT durchgeführten Code-Übersetzungen von COBOL zu Python vordefinierte Qualitätskriterien?“* Dafür wurden 20 COBOL Codebeispiele mit ChatGPT-4 in zwei Durchläufen – einmal mit und einmal ohne Anweisungen – in die Zielsprache Python übersetzt. Die drei Codebeispiele „isnumeric“, „mergesort“ und „unstring“ wiesen derart unterschiedliche Ausgaben gegenüber dem ursprünglichen Code auf oder zeigten gravierende Probleme in der Logik, die nicht nur wegen der Übersetzung aufkamen, sondern weil COBOL und Python konzeptuell so unterschiedlich sind, dass diese Beispiele im vorliegenden Bewertungsraster nicht numerisch bewertet wurden. Sämtlicher Code, der für die Evaluation verwendet wurde, ist in GitHub unter *https://github.com/stuufc/codesamples.git* zugänglich. Die COBOL Codebeispiele sind im Ordner „COBOL Code“ gespeichert. Die jeweiligen Übersetzungen sind im Ordner „Python Code“ in den Unterordnern „Supervised“ und „Unsupervised“ zugänglich.

6.1 Gesamtergebnisse

Die Gesamtergebnisse der Evaluation wurden aus den separat ausgefüllten Bewertungsrastern zusammengetragen und in einer separaten Excel Datei zusammengefasst. In Abbildung 11 (und Anhang A) kann entsprechend entnommen werden, wie viele von maximal 18 erreichbaren Punkten pro übersetztem Codebeispiel erreicht wurden.

Resultate Bewertung von ChatGPT als Codeübersetzungstool															
Bezeichnung Beispiel	Gleichheit der Ausgaben	Vollständigkeit der Übersetzung	Richtigkeit	Funktionalität	Grösse des Programms	Redundanz	Einhaltung von Best Practices	Optimalität	Lesbarkeit	Variablen und Funktionen	Kommentare	Verständlichkeit	Gesamt	Prozentual
addamount_unsup	2	2	2	6	2	1	1	4	2	1	0	3	13	72%
addamount_sup	2	2	2	6	2	1	1	4	2	1	2	5	15	83%
bubblesort_unsup	2	2	2	6	2	2	2	6	2	1	0	3	15	83%
bubblesort_sup	1	2	2	5	2	2	1	5	2	1	2	5	15	83%
flatprice_unsup	2	2	2	6	1	0	1	2	2	2	0	4	12	67%
flatprice_sup	2	2	2	6	1	1	1	3	1	2	2	5	14	78%
forloop_unsup	1	2	2	5	2	2	2	6	1	1	0	2	13	72%
forloop_sup	0	2	2	4	2	2	1	5	2	1	2	5	14	78%
helloworld2_unsup	2	2	2	6	2	2	2	6	2	2	2	6	18	100%
helloworld2_sup	2	2	2	6	2	2	2	6	2	2	2	6	18	100%
isnumeric_unsup														
isnumeric_sup														
jsongenerate_unsup	2	2	2	6	2	2	1	5	2	1	1	4	15	83%
jsongenerate_sup	2	2	2	6	1	2	1	4	2	1	2	5	15	83%
mergesort_unsup														
mergesort_sup														
notenrechner_unsup	0	0	2	2	0	0	0	0	0	0	0	0	2	11%
notenrechner_sup	2	2	1	5	0	0	0	0	0	0	0	0	5	28%
npkcalculator_unsup	2	2	2	6	2	2	1	5	2	2	0	4	15	83%
npkcalculator_sup	2	2	2	6	1	2	0	3	1	2	1	4	13	72%
numval_unsup	1	2	2	5	2	2	1	5	2	1	1	4	14	78%
numval_sup	1	2	2	5	2	2	2	6	2	1	2	5	16	89%
payroll_unsup	2	2	2	6	2	2	1	5	2	2	0	4	15	83%
payroll_sup	2	2	2	6	2	2	2	6	2	2	2	6	18	100%
randombingo_unsup	0	0	0	0	0	1	0	1	0	0	0	0	1	6%
randombingo_sup	2	2	2	6	0	1	0	1	0	1	1	2	9	50%
randomnumber_unsup	2	2	2	6	2	2	1	5	2	1	0	3	14	78%
randomnumber_sup	2	2	2	6	1	2	1	4	2	1	2	5	15	83%
randomsort_unsup	2	2	2	6	2	2	1	5	2	1	1	4	15	83%
randomsort_sup	2	2	2	6	2	2	2	6	2	1	2	5	17	94%
simplecalculator_unsup	2	2	2	6	2	2	1	5	2	1	0	3	14	78%
simplecalculator_sup	1	2	2	5	1	1	1	3	1	1	0	2	10	56%
stringsplit_unsup	2	2	2	6	2	2	1	5	2	1	0	3	14	78%
stringsplit_sup	2	2	2	6	1	1	1	3	1	1	2	4	13	72%
trim_unsup	2	2	2	6	0	0	0	0	0	0	0	0	6	33%
trim_sup	2	2	2	6	0	0	0	0	0	0	1	1	7	39%
unstring_unsup														
unstring_sup														
whileloop_unsup	1	2	2	5	2	2	2	6	2	1	0	3	14	78%
whileloop_sup	1	2	2	5	2	2	2	6	2	1	0	3	14	78%
Durchschnittswerte ohne Anweisung	1.588235294	1.764705882	1.882352941	5.235294118	1.588235294	1.529411765	1.058823529	4.176470588	1.588235294	1.058823529	0.294117647	2.941176471	12.35294118	69%
Durchschnittswerte mit Anweisung	1.647058824	2	1.941176471	5.588235294	1.294117647	1.470588235	1.058823529	3.823529412	1.411764706	1.117647059	1.470588235	4	13.41176471	75%
Durschnittswerte gesamt	1.617647059	1.882352941	1.911764706	5.411764706	1.441176471	1.5	1.058823529	4	1.5	1.088235294	0.882352941	3.470588235	12.88235294	72%

Abbildung 11: Resultate der Bewertung von ChatGPT als Codeübersetzungstool. (eigene Darstellung)

Für das Berechnen der Durchschnittswerte wurden die erreichten Punktzahlen der bewerteten Übersetzungen von 17 Codebeispielen zur Hand genommen und durch 34 geteilt.

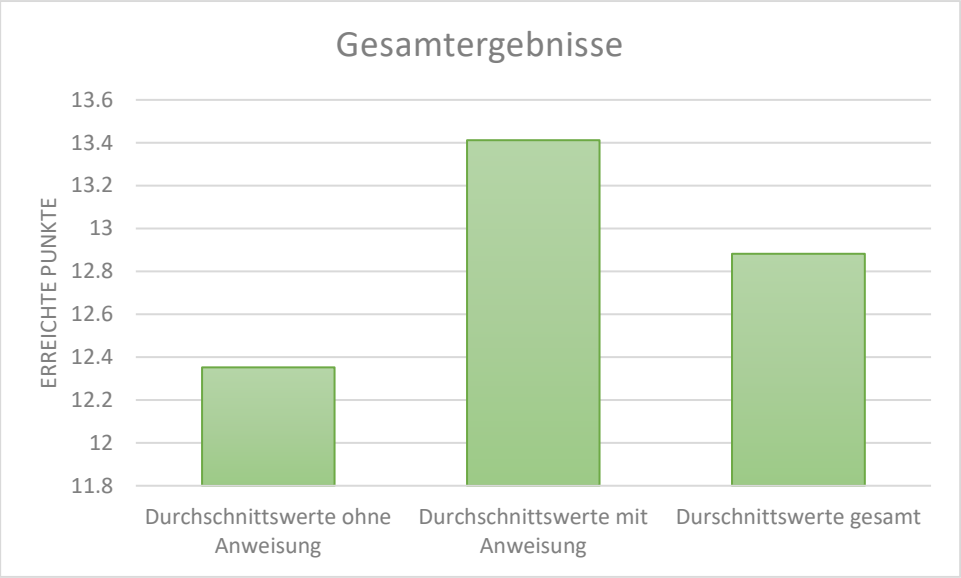


Abbildung 12: Gesamtergebnisse der Evaluation von ChatGPT. (eigene Darstellung)

Die Übersetzungen ohne Anweisung haben im Durchschnitt 12.35 von 18 Punkten erreicht (69 Prozent). Die zweite Durchführung mit Anweisung hat zu einem durchschnittlichen Ergebnis von 13.41 Punkten geführt (75 Prozent). Basierend auf der gewählten Methodik zur Evaluierung erreicht ChatGPT gesamthaft als unterstützendes Tool zur teilautomatisierten Codeübersetzung also 12.88 Punkte im Durchschnitt (72 Prozent).

6.2 Ergebnisse nach Kriterien

In diesem Kapitel werden die Ergebnisse nach Kriterien und Unterkriterien präsentiert.

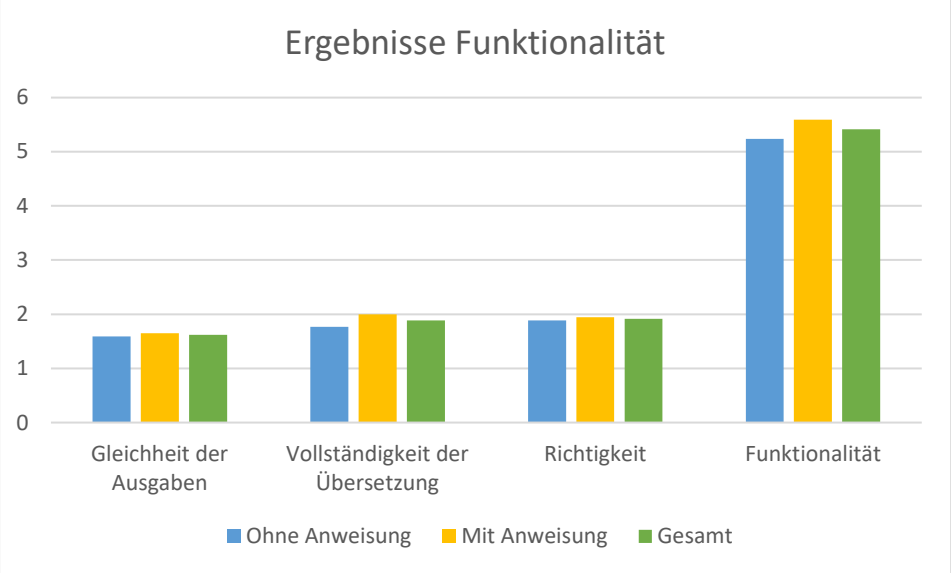


Abbildung 13: Ergebnisse der Evaluation des Kriteriums Funktionalität. (eigene Darstellung)

Für das Kriterium Funktionalität wurden im Gesamten durchschnittlich 5.41 von 6 möglichen Punkten erreicht (siehe Abbildung 13). Im Durchlauf ohne Anweisung wurde ein Wert von 5.24 erreicht, der sich aus 1.59 Punkten bei Gleichheit der Ausgaben, 1.76 Punkten bei Vollständigkeit der Übersetzung und 1.88 Punkten bei Richtigkeit zusammensetzt. Der Durchlauf mit Anweisung schnitt mit 5.59 Punkten im Durchschnitt etwas besser ab. Dieses Ergebnis setzt sich aus 1.65 Punkten bei Gleichheit der Ausgaben, 2 Punkten bei Vollständigkeit der Übersetzung und 1.94 Punkten bei Richtigkeit zusammen.

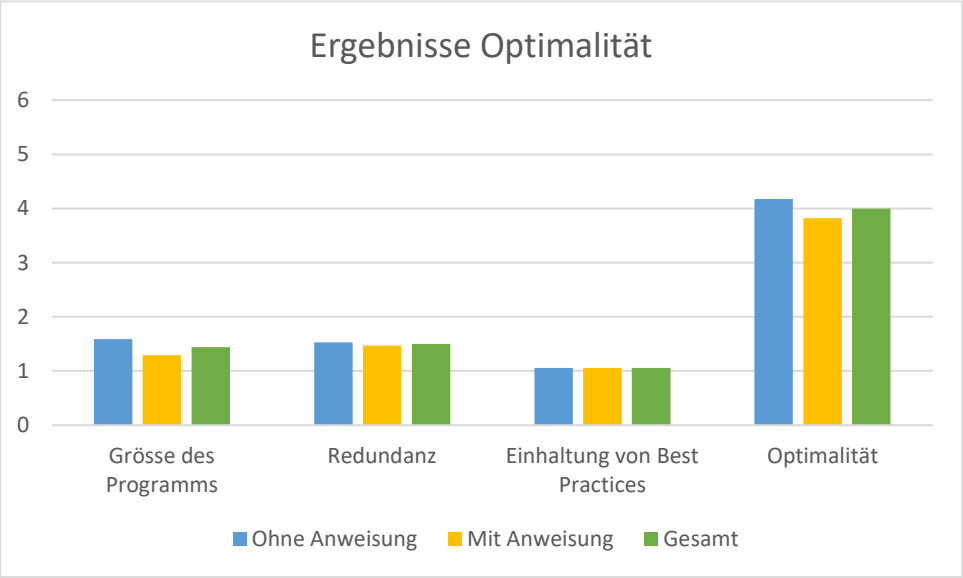


Abbildung 14: Ergebnisse der Evaluation des Kriteriums Optimalität. (eigene Darstellung)

Für das Kriterium Optimalität wurden im Gesamten durchschnittlich 4 von 6 möglichen Punkten erreicht (siehe Abbildung 14). Ohne Anweisungen konnte ChatGPT einen durchschnittlichen Wert von 4.18 erreichen. Dabei wurden für das Unterkriterium Grösse des Programms 1.59 Punkte, für Redundanz 1.53 Punkte und für Einhaltung von Best Practices 1.06 Punkte erreicht. Mit Anweisung erreichte ChatGPT durchschnittlich 3.82 Punkte. Das Kriterium Grösse des Programms wurde durchschnittlich mit 1.29 Punkten, Redundanz mit 1.47 und Einhaltung von Best Practices mit 1.06 bewertet.

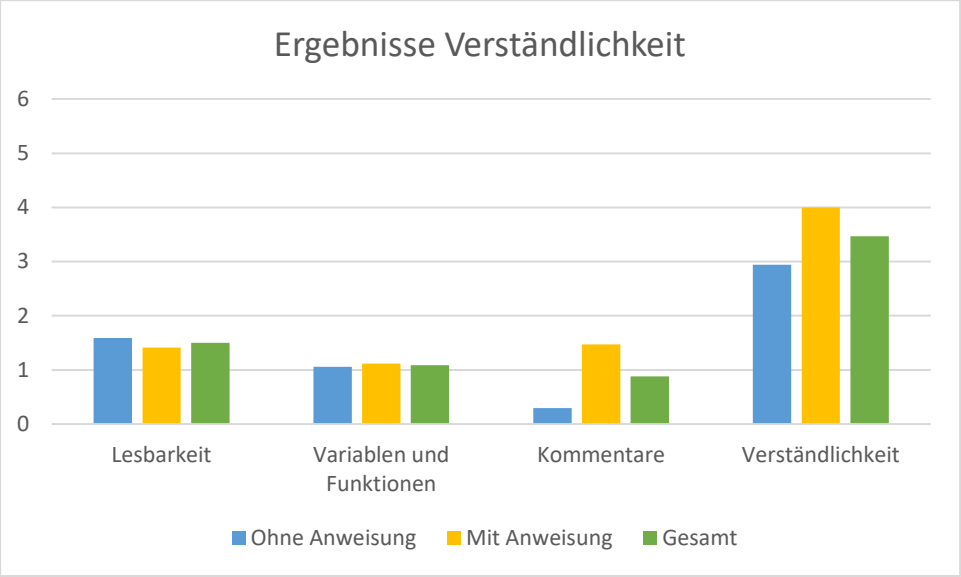


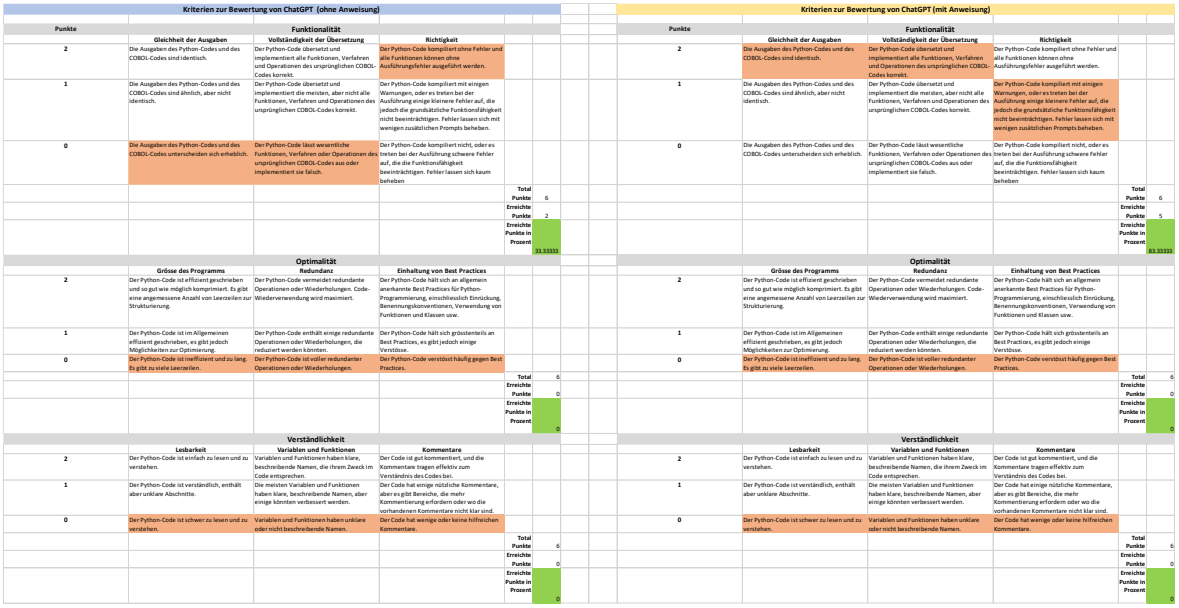
Abbildung 15: Ergebnisse der Evaluation des Kriteriums Verständlichkeit. (eigene Darstellung)

Für das Kriterium Verständlichkeit wurden im Gesamten durchschnittlich 3.47 von 6 möglichen Punkten erreicht (siehe Abbildung 15). Im Durchlauf ohne Anweisungen konnten durchschnittlich 2.94 Punkte und für die Unterkriterien Lesbarkeit 1.59 Punkte, für

Variablen und Funktionen 1.06 Punkte und für Kommentare 0.29 Punkte erreicht werden. Im Durchlauf mit Anweisungen setzt sich der erreichte Durchschnittswert von 4 Punkten aus 1.41 Punkten in Lesbarkeit, 1.12 Punkten in Variablen und Funktionen sowie 1.47 Punkten in Kommentaren zusammen.

6.3 Beispiele Codeübersetzung

In diesem Abschnitt werden exemplarisch ein negatives und ein positives Beispiel präsentiert. Zunächst wurde dafür ein simpler Notenrechner als Negativbeispiel ausgewählt. Die dazugehörige Bewertung ist in Abbildung 16 zu sehen und kann in Anhang B detaillierter betrachtet werden.



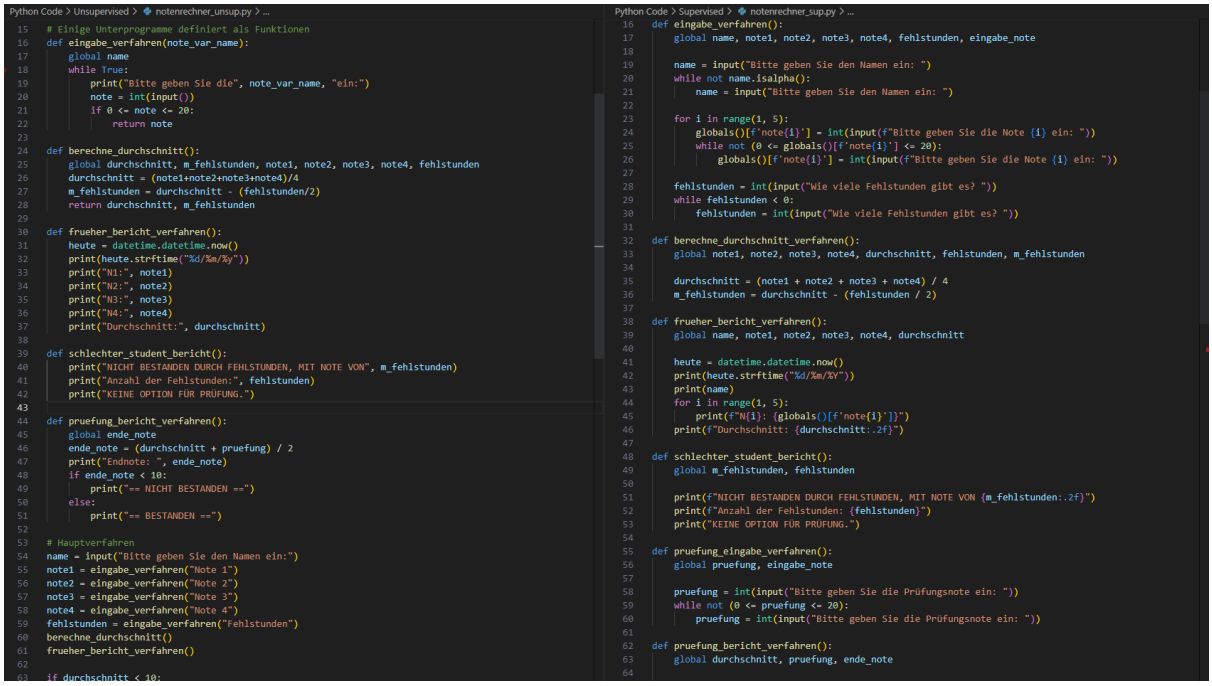
Kriterien zur Bewertung von ChatGPT (ohne Anweisung)					
Punkte	Funktionalität				
	Gleichheit der Ausgaben	Vollständigkeit der Übersetzung	Richtigkeit		
2	Die Ausgaben des Python-Codes und des COBOL-Codes sind identisch.	Der Python-Code übersetzt und implementiert alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert ohne Fehler und alle Funktionen können ohne Ausführungsfehler ausgeführt werden.		
1	Die Ausgaben des Python-Codes und des COBOL-Codes sind ähnlich, aber nicht identisch.	Der Python-Code übersetzt und implementiert die meisten, aber nicht alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert mit einigen Warnungen, oder es treten bei der Ausführung einige kleinere Fehler auf, die jedoch die grundsätzliche Funktionsfähigkeit nicht beeinträchtigen. Fehler lassen sich mit wenigen zusätzlichen Prompts beheben.		
0	Die Ausgaben des Python-Codes und des COBOL-Codes unterscheiden sich erheblich.	Der Python-Code lässt wesentliche Funktionen, Verfahren oder Operationen des ursprünglichen COBOL-Codes aus oder implementiert sie falsch.	Der Python-Code kompiliert nicht, oder es treten bei der Ausführung schwere Fehler auf, die die Funktionsfähigkeit beeinträchtigen. Fehler lassen sich kaum beheben		
				Total Punkte	6
				Erreichte Punkte	2
				Erreichte Punkte in Prozent	33.33333
Optimalität					
	Grösse des Programms	Redundanz	Einhaltung von Best Practices		
2	Der Python-Code ist effizient geschrieben und so gut wie möglich komprimiert. Es gibt eine angemessene Anzahl von Leerzeilen zur Strukturierung.	Der Python-Code vermeidet redundante Operationen oder Wiederholungen. Code-Wiederverwendung wird maximiert.	Der Python-Code hält sich an allgemein anerkannte Best Practices für Python-Programmierung, einschliesslich Einrückung, Benennungskonventionen, Verwendung von Funktionen und Klassen usw.		
1	Der Python-Code ist im Allgemeinen effizient geschrieben, es gibt jedoch Möglichkeiten zur Optimierung.	Der Python-Code enthält einige redundante Operationen oder Wiederholungen, die reduziert werden könnten.	Der Python-Code hält sich grösstenteils an Best Practices, es gibt jedoch einige Verstösse.		
0	Der Python-Code ist ineffizient und zu lang. Es gibt zu viele Leerzeilen.	Der Python-Code ist voller redundanter Operationen oder Wiederholungen.	Der Python-Code verstösst häufig gegen Best Practices.		
				Total Punkte	6
				Erreichte Punkte	0
				Erreichte Punkte in Prozent	0
Verständlichkeit					
	Lesbarkeit	Variablen und Funktionen	Kommentare		
2	Der Python-Code ist einfach zu lesen und zu verstehen.	Variablen und Funktionen haben klare, beschreibende Namen, die ihrem Zweck im Code entsprechen.	Der Code ist gut kommentiert, und die Kommentare tragen effektiv zum Verständnis des Codes bei.		
1	Der Python-Code ist verständlich, enthält aber unklare Abschnitte.	Die meisten Variablen und Funktionen haben klare, beschreibende Namen, aber einige könnten verbessert werden.	Der Code hat einige nützliche Kommentare, aber es gibt Bereiche, die mehr Kommentierung erfordern oder wo die vorhandenen Kommentare nicht klar sind.		
0	Der Python-Code ist schwer zu lesen und zu verstehen.	Variablen und Funktionen haben unklare oder nicht beschreibende Namen.	Der Code hat wenige oder keine hilfreichen Kommentare.		
				Total Punkte	6
				Erreichte Punkte	0
				Erreichte Punkte in Prozent	0

Kriterien zur Bewertung von ChatGPT (mit Anweisung)					
Punkte	Funktionalität				
	Gleichheit der Ausgaben	Vollständigkeit der Übersetzung	Richtigkeit		
2	Die Ausgaben des Python-Codes und des COBOL-Codes sind identisch.	Der Python-Code übersetzt und implementiert alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert ohne Fehler und alle Funktionen können ohne Ausführungsfehler ausgeführt werden.		
1	Die Ausgaben des Python-Codes und des COBOL-Codes sind ähnlich, aber nicht identisch.	Der Python-Code übersetzt und implementiert die meisten, aber nicht alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert mit einigen Warnungen, oder es treten bei der Ausführung einige kleinere Fehler auf, die jedoch die grundsätzliche Funktionsfähigkeit nicht beeinträchtigen. Fehler lassen sich mit wenigen zusätzlichen Prompts beheben.		
0	Die Ausgaben des Python-Codes und des COBOL-Codes unterscheiden sich erheblich.	Der Python-Code lässt wesentliche Funktionen, Verfahren oder Operationen des ursprünglichen COBOL-Codes aus oder implementiert sie falsch.	Der Python-Code kompiliert nicht, oder es treten bei der Ausführung schwere Fehler auf, die die Funktionsfähigkeit beeinträchtigen. Fehler lassen sich kaum beheben		
				Total Punkte	6
				Erreichte Punkte	5
				Erreichte Punkte in Prozent	83.33333
Optimalität					
	Grösse des Programms	Redundanz	Einhaltung von Best Practices		
2	Der Python-Code ist effizient geschrieben und so gut wie möglich komprimiert. Es gibt eine angemessene Anzahl von Leerzeilen zur Strukturierung.	Der Python-Code vermeidet redundante Operationen oder Wiederholungen. Code-Wiederverwendung wird maximiert.	Der Python-Code hält sich an allgemein anerkannte Best Practices für Python-Programmierung, einschliesslich Einrückung, Benennungskonventionen, Verwendung von Funktionen und Klassen usw.		
1	Der Python-Code ist im Allgemeinen effizient geschrieben, es gibt jedoch Möglichkeiten zur Optimierung.	Der Python-Code enthält einige redundante Operationen oder Wiederholungen, die reduziert werden könnten.	Der Python-Code hält sich grösstenteils an Best Practices, es gibt jedoch einige Verstösse.		
0	Der Python-Code ist ineffizient und zu lang. Es gibt zu viele Leerzeilen.	Der Python-Code ist voller redundanter Operationen oder Wiederholungen.	Der Python-Code verstösst häufig gegen Best Practices.		
				Total Punkte	6
				Erreichte Punkte	0
				Erreichte Punkte in Prozent	0
Verständlichkeit					
	Lesbarkeit	Variablen und Funktionen	Kommentare		
2	Der Python-Code ist einfach zu lesen und zu verstehen.	Variablen und Funktionen haben klare, beschreibende Namen, die ihrem Zweck im Code entsprechen.	Der Code ist gut kommentiert, und die Kommentare tragen effektiv zum Verständnis des Codes bei.		
1	Der Python-Code ist verständlich, enthält aber unklare Abschnitte.	Die meisten Variablen und Funktionen haben klare, beschreibende Namen, aber einige könnten verbessert werden.	Der Code hat einige nützliche Kommentare, aber es gibt Bereiche, die mehr Kommentierung erfordern oder wo die vorhandenen Kommentare nicht klar sind.		
0	Der Python-Code ist schwer zu lesen und zu verstehen.	Variablen und Funktionen haben unklare oder nicht beschreibende Namen.	Der Code hat wenige oder keine hilfreichen Kommentare.		
				Total Punkte	6
				Erreichte Punkte	0
				Erreichte Punkte in Prozent	0

Abbildung 16: Bewertung des Codebeispiels Notenrechner ohne Anweisung (links) und mit Anweisung (rechts). (eige-ne Darstellung)

Der Durchlauf ohne Anweisungen (links) erzielte insgesamt nur 2 Punkte und das nur deshalb, weil der Code ausführbar war. Allerdings wurde in sämtlichen anderen Unterkriterien 0 Punkte erreicht, weil keine Best Practices befolgt wurden, der Code andere Ausgaben generierte und globale Variablen verwendet wurden. Das würde gerade bei Verwendung eines Scripts in einem grösseren Programm zu Problemen führen. Dazu kommen ineffiziente if-Abfragen und grundsätzlich ein Code, der viel zu lange ist. Insgesamt umfasst die Ausgabe ohne Anleitung 73 Zeilen Code, Kommentare und Leerzeilen. Die Ausgabe mit Anweisungen wies zunächst Probleme beim Kompilieren auf, die jedoch mit einem weiteren Prompt behoben werden konnten. Danach zeigten sich ähnliche Probleme, da der Code nach wie vor globale Variablen verwendete, keine sinnvolle Struktur verwendet wurde und der Code grundsätzlich sehr kompliziert zu verstehen war.

Trotz Anweisung wurden keine Kommentare in den Code eingefügt. Da der Code immerhin die gleichen Ausgaben erzielte, erhielt das mit Anweisungen übersetzte Beispiel insgesamt 5 von 18 Punkten. In Abbildung 17 sind Ausschnitte des übersetzten Codes, links ohne und rechts mit Anweisung zu sehen.



```
Python Code > Unsupervised > notenrechner_unsup.py > ...
# Einige Unterprogramme definiert als Funktionen
def eingabe_verfahren(note_var_name):
    global name
    while True:
        print("Bitte geben Sie die", note_var_name, "ein:")
        note = int(input())
        if 0 <= note <= 20:
            return note

def berechne_durchschnitt():
    global durchschnitt, m_fehlstunden, note1, note2, note3, note4, fehlstunden
    durchschnitt = (note1+note2+note3+note4)/4
    m_fehlstunden = durchschnitt - (fehlstunden/2)
    return durchschnitt, m_fehlstunden

def frueher_bericht_verfahren():
    heute = datetime.datetime.now()
    print(heute.strftime("%d/%m/%y"))
    print("N1:", note1)
    print("N2:", note2)
    print("N3:", note3)
    print("N4:", note4)
    print("Durchschnitt:", durchschnitt)

def schlechter_student_bericht():
    print("NICHT BESTANDEN DURCH FEHLSTUNDEN, MIT NOTE VON", m_fehlstunden)
    print("Anzahl der Fehlstunden:", fehlstunden)
    print("KEINE OPTION FÜR PRÜFUNG.")

def pruefung_bericht_verfahren():
    global ende_note
    ende_note = (durchschnitt + pruefung) / 2
    print("Endnote: ", ende_note)
    if ende_note < 10:
        print("== NICHT BESTANDEN ==")
    else:
        print("== BESTANDEN ==")

# Hauptverfahren
name = input("Bitte geben Sie den Namen ein:")
note1 = eingabe_verfahren("Note 1")
note2 = eingabe_verfahren("Note 2")
note3 = eingabe_verfahren("Note 3")
note4 = eingabe_verfahren("Note 4")
fehlstunden = eingabe_verfahren("Fehlstunden")
berechne_durchschnitt()
frueher_bericht_verfahren()

if durchschnitt < 10:
```

```
Python Code > Supervised > notenrechner_sup.py > ...
def eingabe_verfahren():
    global name, note1, note2, note3, note4, fehlstunden, eingabe_note

    name = input("Bitte geben Sie den Namen ein: ")
    while not name.isalpha():
        name = input("Bitte geben Sie den Namen ein: ")

    for i in range(1, 5):
        globals()[f'note{i}'] = int(input(f"Bitte geben Sie die Note {i} ein: "))
        while not (0 <= globals()[f'note{i}'] <= 20):
            globals()[f'note{i}'] = int(input(f"Bitte geben Sie die Note {i} ein: "))

    fehlstunden = int(input("Wie viele Fehlstunden gibt es? "))
    while fehlstunden < 0:
        fehlstunden = int(input("Wie viele Fehlstunden gibt es? "))

def berechne_durchschnitt_verfahren():
    global note1, note2, note3, note4, durchschnitt, fehlstunden, m_fehlstunden

    durchschnitt = (note1 + note2 + note3 + note4) / 4
    m_fehlstunden = durchschnitt - (fehlstunden / 2)

def frueher_bericht_verfahren():
    global name, note1, note2, note3, note4, durchschnitt

    heute = datetime.datetime.now()
    print(heute.strftime("%d/%m/%Y"))
    print(name)
    for i in range(1, 5):
        print(f"N{i}: {globals()[f'note{i}']}")
    print(f"Durchschnitt: {durchschnitt:.2f}")

def schlechter_student_bericht():
    global m_fehlstunden, fehlstunden

    print(f"NICHT BESTANDEN DURCH FEHLSTUNDEN, MIT NOTE VON {m_fehlstunden:.2f}")
    print(f"Anzahl der Fehlstunden: {fehlstunden}")
    print("KEINE OPTION FÜR PRÜFUNG.")

def pruefung_eingabe_verfahren():
    global pruefung, eingabe_note

    pruefung = int(input("Bitte geben Sie die Prüfungsnote ein: "))
    while not (0 <= pruefung <= 20):
        pruefung = int(input("Bitte geben Sie die Prüfungsnote ein: "))

def pruefung_bericht_verfahren():
    global durchschnitt, pruefung, ende_note
```

Abbildung 17: Übersetzung des Notenrechners ohne (links) und mit (rechts) Anweisung. (eigene Darstellung)

Wenn man ChatGPT nicht die Anweisung gibt, den Notenrechner zu übersetzen, sondern ein Programm mit einer vergleichbaren Funktionalität zu generieren, dann generiert ChatGPT deutlich effizienteren und verständlicheren Code (siehe Abbildung 18):

```python
def calculate_average(grades):
    return sum(grades) / len(grades)

def check_pass_status(grades, absent_hours):
    if absent_hours > 20:
        return "Nicht bestanden wegen zu vieler Fehlstunden."
    elif calculate_average(grades) <= 10:
        return "Nicht bestanden wegen zu niedrigem Notendurchschnitt."
    else:
        return "Bestanden."

def main():
    grades = []
    for i in range(1, 5):
        grade = float(input(f"Geben Sie Note {i} ein: "))
        grades.append(grade)
    absent_hours = int(input("Geben Sie die Anzahl der Fehlstunden ein: "))
    print(check_pass_status(grades, absent_hours))

if __name__ == "__main__":
    main()
```

Abbildung 18: Von ChatGPT generierter Notenrechner. (eigene Darstellung)

Im Vergleich zum Notenrechner konnte ChatGPT ein fast perfektes Ergebnis für das Beispiel „randomsort“ erzielen (siehe Abbildung 19). Nur die Benennung von Variablen und Funktionen hätte optimiert werden können, da diese aus COBOL übernommen wurden und so wenig aussagekräftig waren. Im Beispiel ohne Anleitung fehlte zudem die Main Methode, welche im Falle des Einbettens des Codes in ein Drittprogramm relevant wäre.

Python Code > Unsupervised > randomsort_unsup.py > ...

```python
import random

def main():
    w_len_arr = 20
    w_arr = [0] * w_len_arr

    # Generiere Zufallszahlen und fülle das Array
    for i in range(w_len_arr):
        w_ran_num = random.randint(1, 99)
        w_arr[i] = w_ran_num
        print("POS: ", i+1, " RANDOM NUMBER: ", w_arr[i])

    # Sortiere das Array (Einfüge-Sortierung)
    for i in range(1, w_len_arr):
        key = w_arr[i]
        j = i - 1

        while j >=0 and key < w_arr[j] :
                w_arr[j+1] = w_arr[j]
                j -= 1

        w_arr[j+1] = key

    # Ausgabe des sortierten Arrays
    for i in range(w_len_arr):
        print("POS: ", i+1, " SORTED: ", w_arr[i])

main()
```

Python Code > Supervised > randomsort_sup.py > ...

```python
import random

def main():
    # Initialisieren der Variablen
    w_len_arr = 20
    w_arr = [0] * w_len_arr

    # Generieren von zufälligen Zahlen
    for w_r in range(w_len_arr):
        w_ran_num = random.randint(1, 99)  # Erzeugen einer Zufallszahl zwischen 1 und 99
        w_arr[w_r] = w_ran_num
        print("POS: ", w_r + 1, " RANDOM NUMBER: ", w_arr[w_r])

    # Sortieren des Arrays mit dem Insertion-Sort-Algorithmus
    for w_i in range(1, w_len_arr):
        w_key = w_arr[w_i]
        w_j = w_i - 1

        while w_j >= 0 and w_arr[w_j] > w_key:
            w_arr[w_j + 1] = w_arr[w_j]
            w_j -= 1

        w_arr[w_j + 1] = w_key

    # Anzeigen der sortierten Elemente
    for w_h in range(w_len_arr):
        print("POS: ", w_h + 1, " SORTED: ", w_arr[w_h])

if __name__ == "__main__":
    main()
```

Abbildung 19: Übersetzung von „randomsort“ ohne (links) und mit (rechts) Anweisung. (eigene Darstellung)

6.4 Zusammenfassung

Die Evaluation von ChatGPT-4 als Tool zur teilautomatisierten Übersetzung von COBOL Code zu Python Code hat aufgezeigt, dass die Qualität der erzeugten Übersetzungen basierend auf den verwendeten Kriterien durchschnittlich bei rund 72 Prozent liegt, wobei

das Ergebnis mit Anweisungen (75 Prozent) 6 Prozentpunkte besser ist als ohne Anweisungen (69 Prozent). Damit kann die dritte Forschungsunterfrage „*Inwiefern erfüllen die von ChatGPT durchgeführten Code-Übersetzungen von COBOL zu Python vordefinierte Qualitätskriterien?“* beantwortet werden. Der Unterschied von 6 Prozentpunkten lässt sich damit erklären, dass im Durchlauf mit Anweisungen vermehrt Kommentare zum Code hinzugefügt wurden, was zu einer besseren Verständlichkeit beigetragen hat. Die Werte beim Kriterium Funktionalität weichen hingegen nur marginal voneinander ab. Überraschenderweise erreichte der Durchlauf ohne Anweisungen bessere Werte für das Kriterium Optimalität. Insgesamt konnten drei von 20 Codebeispielen für die detaillierte Bewertung nicht berücksichtigt werden, da entweder der generierte Code nicht komplett war, die COBOL Logik nicht passend übertragen werden konnte oder die Outputs der Programme zu unterschiedlich waren. Ansonsten lässt sich festhalten, dass nur die Beispiele des „Hello World“ Programms, sowie die angeleitete Version des Beispiels „payroll“ ein perfektes Übersetzungsergebnis lieferten. Die meisten Übersetzungen wurden mit 12 bis 15 Punkten bewertet und wiesen damit Verbesserungspotenzial auf. Dieses Ergebnis wurde durch positive und negative Ausreisser (siehe Kapitel 6.3) beeinflusst. Sämtliche Codebeispiele in COBOL und die übersetzten Python Codes können über *https://github.com/stuufc/codesamples.git* abgerufen und verglichen werden. Das detaillierte Bewertungsraster pro Codebeispiel kann aus Anhang B entnommen werden.

7 Diskussion

Ziel dieser Bachelorarbeit war es herauszufinden, wie gut sich ChatGPT basierend auf dem GPT-4 Modell als unterstützendes Tool zur Migration von Programmiersprachen eignet, was für Potenziale mit der Verwendung einhergehen und was für Herausforderungen es zu überwinden gibt. In diesem Kapitel werden die Ergebnisse aus Kapitel 6 interpretiert und diskutiert. Basierend darauf werden die Forschungsfragen beantwortet, auf Einschränkungen der Evaluation und entsprechende Vorschläge für zukünftige Forschung eingegangen und Implikationen für die Praxis erwähnt.

7.1 Interpretation der Ergebnisse

Die Bewertung von ChatGPT wurde mit einfachen Codebeispielen durchgeführt. Dennoch lassen sich daraus Erkenntnisse ableiten, die bei komplexerem Code mit grosser Wahrscheinlichkeit auch zutreffen.

Die Haupterkenntnis der durchgeführten Evaluation ist, dass ChatGPT nicht die kognitiven Fähigkeiten besitzt, Code tatsächlich zu verstehen und optimiert in einer Zielsprache auszugeben. Diese Erkenntnis geht unter anderem mit Borji (2023) einher, der ChatGPT Probleme in den Bereichen Logik und Argumentationsfähigkeit zuschreibt. ChatGPT nimmt Code zusammen mit einer Textanweisung in natürlicher Sprache entgegen, analysiert den Code und versucht dann, den Code womöglich 1 zu 1 zu übersetzen. Das in Kapitel 6.3 aufgezeigte Negativbeispiel unterstreicht dieses Verhalten. ChatGPT ist nicht in der Lage, den kognitiv anspruchsvollen Teil, nämlich das COBOL Programm zu verstehen und zu beschreiben, was es in der Zielsprache machen soll, zu bewältigen. Ausserdem fehlt ChatGPT die Fähigkeit von selbst zu adaptieren und optimale Programmierparadigmen der Zielsprache anzuwenden. Stattdessen versucht ChatGPT, die Struktur des Ausgangscodes zu imitieren und möglichst identisch in der Zielsprache abzubilden. Das zeigt sich auch durch das Übernehmen von Variablen- und Funktionsnamen trotz geringer Aussagekraft. Dies führt dazu, dass der Code mitunter nicht einfach zu lesen ist und gängige Praktiken des Öfteren missachtet.

Eine weitere Erkenntnis ist die Zufälligkeit der Ausgaben, die so auch von Tian et al. (2023) beobachtet wurde. Diese Zufälligkeit wurde vor allem durch das Unterkriterium „Kommentare“, die Verwendung einer Main-Methode und bei der Verwendung der random-Funktionalität in Python ersichtlich. Ohne Anweisung wurden beispielsweise in 4 von 17 Beispielen Kommentare hinzugefügt, während trotz Anweisung Kommentare einzufügen 3 von 17 Beispiele keine Kommentare enthielten. Des Weiteren wurde Code

nicht immer so geschrieben, dass er sich für eine Wiederverwendung in einem grösseren Programm eignet. Es konnte kein Muster erkannt werden, wann und warum ChatGPT eine Main-Methode in den Code einfügte und diesen für die Wiederverwendung optimierte und wann nicht. Eine weitere Unstimmigkeit im Kontext der Zufälligkeit zeigte sich beim Beispiel „randombingo“. Hier wurde die random-Funktion im Vergleich zu den Beispielen „randomnumber“ und „randomsort“ umständlich und inkorrekt verwendet. Anstatt einfach eine Zufallszahl zwischen 1 und 100 auszugeben, wurde der Code so geschrieben, dass die Zufallszahl 200 Mal generiert und der 200. Wert ausgegeben wird.

Noch eine abgeleitete Erkenntnis ist, dass die Qualität von Ausgaben eines LLMs durch den Einsatz von Prompt Engineering beeinflusst werden kann. Szafraniec et al. (2023) kamen auf eine Verbesserung der Übersetzungen bezogen auf deren Richtigkeit von 11 Prozent, wenn dem verwendeten Tool zusätzliche Anweisungen mitgegeben wurden. In dieser Arbeit wurde eine Verbesserung des Ergebnisses von 6 Prozentpunkten unter Einsatz von Prompt Engineering festgestellt. Es ist davon auszugehen, dass ein zielgerichtetes Prompt Engineering diesen Wert noch einmal deutlich anheben kann. Im Rahmen dieser Arbeit wurde nämlich für jedes Beispiel derselbe Prompt verwendet. Dieser allgemeine Prompt hat keine detaillierten Angaben zum Kontext eines Codes beinhaltet.

Bis auf wenige Ausreisser wurden die meisten Übersetzungen mit 12 bis 15 Punkten bewertet. Das lässt darauf schliessen, dass viele der Übersetzungen grundsätzlich gut, nicht aber perfekt sind. Das deckt sich mit den Erkenntnissen von Vaithilingam et al. (2022) und Moradi Dakhel et al. (2022), die sagen, dass KI-Modelle nicht immer optimale Ausgaben generieren, diese aber ein guter Startpunkt für die Weiterverwendung sein können. ChatGPT hat es mehrheitlich geschafft, Code so zu übersetzen, dass bei gleichem Input gleiche Ausgaben generiert werden. In diversen Beispielen, darunter dem „simplecalculator“ kann der Code dann einfach manuell oder mit Hilfe von ChatGPT optimiert werden. Einfache Optimierungen, die mehrheitlich nötig gewesen wären, sind beispielsweise das Verwenden von Loops für sich wiederholende Operationen (z. B. print-Statements) oder das Verwenden von klareren Variablen- und Funktionsnamen.

7.2 Beantwortung der Forschungsfragen

Anhand von drei Forschungsunterfragen wurde in dieser Arbeit darauf hingearbeitet, folgende forschungsleitende Fragestellung zu beantworten: *„Welche Potenziale und Limitationen weist ChatGPT als unterstützendes Tool für die Code-Migration von COBOL zu Python auf?“*. Zunächst wurde untersucht, was für Potenziale und Herausforderungen die Anwendung von generativer KI im Bereich der Programmierung mit sich bringt.

Generative KI soll vor allem die Effizienz von Programmierer:innen steigern und daher Kosteneinsparungen mit sich bringen und die Genauigkeit in verschiedenen Programmieraufgaben verbessern. Generell geht damit ein enormes ökonomisches Potenzial einher, welches durch immer besser werdende Modelle, verbesserte Trainingsdaten und die Anpassung der Modelle auf bestimmte Anwendungsfälle weiter ausgeschöpft werden kann. Zu den grössten Herausforderungen gehören das Vertrauen in KI-Tools, die Nutzungsfreundlichkeit, Voreingenommenheit des Modells, rechtliche und regulatorische Themen sowie das Erkennen von Zusammenhängen, logisches Denken und generell der Missbrauch von KI-Tools für schädliche Zwecke. In Folge wurden anhand einer weitläufigen Literaturrecherche Kriterien abgeleitet, anhand welcher die Leistungsfähigkeit von ChatGPT im Bereich der Codemigration bzw. Codeübersetzung bewertet werden kann. Das Resultat dieser Erhebung ist das in Kapitel 5.3 präsentierte Modell zur Bewertung der Ausgabequalität von ChatGPT. Die Evaluation wurde anhand der Hauptkriterien Funktionalität, Optimalität und Verständlichkeit durchgeführt und hat zu dem Ergebnis geführt, dass ChatGPT die Übersetzungen basierend auf den erarbeiteten Kriterien mit 72 Prozent Erfolg übersetzt hat.

Die Erkenntnisse aus den Forschungsunterfragen und der Evaluation von ChatGPT erlauben die Beantwortung der forschungsleitenden Fragestellung. Durch das Design von ChatGPT, das im Stile eines Chatbots natürliche Sprache versteht, wurde die Voraussetzung geschaffen, dass das Tool weitläufig Verwendung findet. ChatGPT erlaubt damit einen einfachen Einstieg für Nutzer:innen, die das Tool so für ihren Anwendungsfall ausprobieren und verwenden können. Die Evaluation hat aufgezeigt, dass ChatGPT hauptsächlich als unterstützendes Tool zur Codemigration betrachtet werden muss. ChatGPT hat jedoch das Potenzial, Entwickler:innen in einer beratenden Funktion zur Seite zu stehen, indem das Tool die Bedeutung und die Struktur des Codes erklärt. Dies ist insbesondere für die Anwendung auf veraltete Programmiersprachen hilfreich, da oft das Verständnis für jene Sprachen fehlt. ChatGPT ist in den meisten Fällen in der Lage, Code zu generieren, welcher die Funktionalität der Ausgangssprache abbildet. Für einfache Beispiele hat ChatGPT diesbezüglich gute Resultate erzielt. Für komplexeren Code bietet sich die Anwendung von „Prompt Engineering“ an. Mit Hilfe dieses Ansatzes und der Fähigkeit von ChatGPT, Fehler im Code effizient zu erkennen und hervorzuheben, können mit dem Tool deutlich bessere Ergebnisse in verschiedenen Aspekten erreicht werden, was letztlich zu einer erhöhten Codequalität führt. Bei sehr unterschiedlichen Programmiersprachen wie COBOL und Python bietet es sich ausserdem an, lediglich einzelne Codeausschnitte oder Funktionen in der Zielsprache generieren zu lassen, und ein Programm, das auf der Struktur der Ausgangssprache basiert, mithilfe der Anweisungen

von ChatGPT selbst zusammenzubauen. Diese Erkenntnis beruht darauf, dass ChatGPT bei längerem, komplexerem Code (Beispiel mergesort) nicht in der Lage war, ein voll funktionales Programm in der Zielsprache auszugeben. Wird dieses Vorgehen iterativ angewandt und der Code so laufend optimiert, kann auf diese Weise der grössten Limitation von ChatGPT als unterstützendem Tool für die Codemigration begegnet werden. Im Anwendungsfall der Übersetzung versucht ChatGPT nämlich Code nachzubauen, anstatt ihn in der Zielsprache zu optimieren. Dies führt in komplexeren Beispielen zu unleserlichem, kompliziertem Code oder sogar falschem Code. ChatGPT fehlt also die kognitive Fähigkeit, Code wirklich zu verstehen und komplexere Code-Artefakte gezielt zu übersetzen. Übersetzter Code sollte manuell geprüft und getestet werden, da sich gezeigt hat, dass ChatGPT mitunter gegen gängige Konzepte verstösst, indem es beispielsweise globale Variablen für einfache Funktionen verwendet, welche im grösseren Kontext für Probleme sorgen könnten. Des Weiteren limitieren unterschiedliche Programmierparadigmen die Leistungsfähigkeit von ChatGPT. Mitunter können COBOL Konzepte nicht entsprechend in Python abgebildet werden. ChatGPT hat dies stellenweise versucht und so Code produziert, der eigentlich keinen Sinn macht. Schlussendlich limitieren auch die Zufälligkeit der Ausgaben, sowie eine teilweise mangelnde Vollständigkeit. Beide Faktoren können das Vertrauen von Entwickler:innen in das Tool einschränken und so die Anwendung in der Praxis verhindern. Wird das GPT-4 Modell über das Web-Interface verwendet, ist die Kapazität des Modells ein weiterer limitierender Faktor. Im Bearbeitungszeitraum dieser Arbeit war das GPT-4-Modell auf 25 Anfragen alle drei Stunden begrenzt. Zusammenfassend lässt sich sagen, dass ChatGPT im Prozess der Codemigration von COBOL zu Python ein hilfreiches Tool sein kann. Gerade in Bezug auf die Bereitstellung von Informationen, das Erklären von Konzepten und Strukturen sowie Teilübersetzungen kann ChatGPT Entwickler:innen einen Mehrwert bringen. Effizienzsteigerung und verbesserte Codequalität sind weitere Punkte, welche die Anwendung von ChatGPT mit sich bringen kann. ChatGPT darf jedoch nicht als perfektes Tool zur Codemigration angesehen werden und kann erfahrene Entwickler:innen nicht vollständig ersetzen.

7.3 Implikationen für die Praxis

Die vorliegende Arbeit zeigt einen strukturierten Ansatz auf, wie die Qualität von teilautomatisierten Codeübersetzungen bewertet und durchgeführt werden kann. Wie in Abschnitt 7.4 beschrieben wird, kann das Bewertungsmodell mit Anpassungen durchaus für die Bewertung von Code- oder Codeübersetzungsqualität verwendet werden. Um effektiv

einschätzen zu können, inwiefern ChatGPT Codemigrationen in der Praxis unterstützen kann, müsste die Forschung wie in Abschnitt 7.4 vorgeschlagen, erweitert werden.

7.4 Vorschläge für zukünftige Forschung

Die Evaluation von ChatGPT zeigt, dass Potenzial in der Nutzung von generativer KI in der Programmierung generell, aber auch in der Migration von einer Programmiersprache in die andere vorhanden ist. Die durchgeführte Untersuchung darf dabei jedoch nicht als abschliessend betrachtet werden. Einerseits ist die Aussagekraft der Evaluation durch die abgeleiteten Kriterien und die Auswahl der Codebeispiele eingeschränkt. Das Bewertungsmodell wurde passend für in sich geschlossene, einfache Codebeispiele, die Grundfunktionalitäten einer Programmiersprache abbilden, aufgestellt. Die abgeleiteten Kriterien wurden anschliessend dafür verwendet, den übersetzten Code ähnlich wie in einem Code Reviews zu bewerten. Für mehr Aussagekraft bezüglich tatsächlicher Adaption von ChatGPT als unterstützendes Codemigrationstool könnten Codeartefakte aus der Praxis, beispielsweise aus Banksoftware für die Tests verwendet werden. Das Bewertungsmodell wurde so aufgestellt, dass es sich für derartige Evaluationen einfach um Kriterien wie Sicherheit, Effizienz oder Komplexität erweitern liesse. Die Verwendung einer grösseren Stichprobe und praxisnäherer Beispiele würde es zudem ermöglichen, die in Kapitel 5 beschriebene Metriken, also quantifizierbare Messgrössen zur Bewertung zu verwenden und so die Evaluation zu verfeinern. Dieser Aspekt wurde aufgrund der eher kleinen Stichprobe in der durchgeführten Erhebung nicht berücksichtigt. Weiter könnten qualitative Interviews mit Expert:innen Aufschluss darüber geben, ob und was für zusätzliche Funktionalitäten ChatGPT mit sich bringen müsste, um von ihnen in der Praxis berücksichtigt zu werden.

Die Leistungsfähigkeit von ChatGPT für die Codemigration wurde in dieser Arbeit anhand des Sprachpaars COBOL und Python untersucht. Dabei wurden Probleme in den Übersetzungen durch strukturelle und konzeptionelle Unterschiede festgestellt. Es würde sich daher anbieten, eine zusätzliche Evaluation anhand eines anderen Sprachpaares durchzuführen. Programmiersprachen, deren Paradigmen und Konzepte sich ähnlicher sind, könnten zu anderen Erkenntnissen in der Übersetzungsevaluation führen.

8 Fazit

In dieser Arbeit wurde der Frage nachgegangen, was für Potenziale und Limitationen ChatGPT (GPT-4) im Anwendungsgebiet der Codemigration bzw. -übersetzung mit sich bringt. Dafür wurde basierend auf einer systematischen Literaturrecherche ein Bewertungsmodell abgeleitet, anhand dessen die Evaluation von ChatGPT durchgeführt werden konnte. Während die Aussagekraft der Evaluation beschränkt ist und weitere Forschungsansätze verfolgt werden müssten, bietet das Bewertungsmodell Potenzial, in der Praxis Anwendung zu finden. Letztlich konnten im Rahmen dieser Arbeit Potenziale und Limitationen lediglich abgewogen werden, um so zum Schluss zu kommen, dass ChatGPT Potenzial in einer unterstützenden Funktion im Prozess der Codemigration mit sich bringt.

Die Auseinandersetzung mit generativer KI im Kontext der Programmierung hat aufgezeigt, dass dieser eine grosse Bedeutung zugeschrieben wird. In der bisherigen Forschung wurde dabei vermehrt auf GitHub Copilot als Tool zur Codegenerierung fokussiert, wobei vor allem die Leistungsfähigkeit fokussiert wurde. Aber auch ethische Themen rund um KI rücken in den Fokus. Gerade seit der Veröffentlichung von ChatGPT und der rasant angestiegenen Zahl an Nutzer:innen werden derartige Randthemen vermehrt betrachtet. Die Forschung über ChatGPT hat derweil ebenfalls an Fahrt aufgenommen. Wie einer interaktiven Grafik von Liu et al. (2023) zu entnehmen ist, wurden Stand 3. Juli 2023 bereits 803 Arbeiten zu ChatGPT erfasst. Die steigende Relevanz der Thematik in der Forschung und die gleichzeitige Weiterentwicklung, Verbesserung und Spezialisierung der zu Grunde liegenden LLMs lässt erahnen, dass generative KI-Modelle in den kommenden Jahren zunehmend an Relevanz gewinnen und in der Praxis verbreiteter eingesetzt werden.

Die Untersuchungen im Bereich der Codemigration und -übersetzung haben sich als vielschichtig erwiesen. Während im Grunde genommen lediglich überprüft werden kann, ob Code in der Zielsprache funktioniert und das Gleiche macht, muss für die praktische Anwendung überprüft werden, ob der Code vordefinierten Qualitätskriterien entspricht. So kann strukturiert überprüft werden, ob der resultierende Code verwendet werden kann. Die Idee der Unterstützung des Migrationsprozesses durch Tools reicht weit zurück und soll dabei helfen, typische Herausforderungen bei Codemigrationen zu überwinden. Dazu gehören hohe Aufwände, mangelnde Expertise und Fehleranfälligkeit. Diese Herausforderungen können mit Tools wie ChatGPT unterstützend angegangen und die Qualität der Lieferobjekte verbessert werden.

9 Literaturverzeichnis

AlOmar, E. A., Mkaouer, M. W., Ouni, A., & Kessentini, M. (2019). On the Impact of Refactoring on the Relationship between Quality Attributes and Design Metrics. *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 1–11. https://doi.org/10.1109/ESEM.2019.8870177

Baggen, R., Correia, J. P., Schill, K., & Visser, J. (2012). Standardized code quality benchmarking for improving software maintainability. *Software Quality Journal*, *20*(2), 287–307. https://doi.org/10.1007/s11219-011-9144-9

Bisbal, J., Lawless, D., Wu, B., Grimson, J., Wade, V., Richardson, R., & O'Sullivan, D. (1997). An overview of legacy information system migration. *Proceedings of Joint 4th International Computer Science Conference and 4th Asia Pacific Software Engineering Conference*, 529–530. https://doi.org/10.1109/APSEC.1997.640219

Biswas, S. (2023). Role of ChatGPT in Computer Programming.: ChatGPT in Computer Programming. *Mesopotamian Journal of Computer Science*, *2023*, 8–16. https://doi.org/10.58496/MJCSC/2023/002

Borji, A. (2023). *A Categorical Archive of ChatGPT Failures* (arXiv:2302.03494). arXiv. http://arxiv.org/abs/2302.03494

Breuker, D. M., Derriks, J., & Brunekreef, J. (2011). Measuring static quality of student code. *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education*, 13–17. https://doi.org/10.1145/1999747.1999754

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., 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., Wu, J., Winter, C., … Amodei, D. (2020). Language Models are Few-Shot Learners. *Advances in Neural Information Processing Systems*, *33*, 1877–1901. https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html

Buse, R. P. L., & Weimer, W. R. (2010). Learning a Metric for Code Readability. *IEEE Transactions on Software Engineering*, *36*(4), 546–558. https://doi.org/10.1109/TSE.2009.70

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. de O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., … Zaremba, W. (2021).

Evaluating Large Language Models Trained on Code (arXiv:2107.03374). arXiv. https://doi.org/10.48550/arXiv.2107.03374

Chisolm, K. C., & Lisonbee, J. C. (1999). The use of computer language compilers in legacy code migration. *1999 IEEE AUTOTESTCON Proceedings (Cat. No.99CH36323)*, 137–145. https://doi.org/10.1109/AUTEST.1999.800370

Chui, M., Roberts, R., Yee, L., Hazan, E., Singla, A., Smaje, K., Sukharevsky, A., & Zemmel, R. (2023, Juni 14). *Economic potential of generative AI | McKinsey*. McKinsey Digital. https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/the-economic-potential-of-generative-ai-the-next-productivity-frontier#key-insights

Dalla Palma, S., Di Nucci, D., Palomba, F., & Tamburri, D. A. (2020). Toward a catalog of software quality metrics for infrastructure code. *Journal of Systems and Software*, *170*, 110726. https://doi.org/10.1016/j.jss.2020.110726

Deng, J., & Lin, Y. (2022). The Benefits and Challenges of ChatGPT: An Overview. *Frontiers in Computing and Intelligent Systems*, *2*(2), Article 2. https://doi.org/10.54097/fcis.v2i2.4465

Ernst, N. A., & Bavota, G. (2022). AI-Driven Development Is Here: Should You Worry? *IEEE Software*, *39*(2), 106–110. https://doi.org/10.1109/MS.2021.3133805

Field, J., & Ramalingam, G. (1999). Identifying procedural structure in Cobol programs. *Proceedings of the 1999 ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering*, 1–10. https://doi.org/10.1145/316158.316163

Geng, C., Zhang, Y., Pientka, B., & Si, X. (2023). *Can ChatGPT Pass An Introductory Level Functional Language Programming Course?* (arXiv:2305.02230). arXiv. http://arxiv.org/abs/2305.02230

Géron, A. (2022). Hands-On Machine Learning With Scikit-Learn And Tensorflow: Concepts, Tools, And Techniques To Build Intelligent Systems—Aurélien Géron. *O'Reilly Media, Inc.*, *Third Edition*.

Gimnich, R., & Winter, A. (2005). Workflows der Software-Migration. *Softwaretechnik-Trends*, 25(2), 22-24.

Haleem, A., Javaid, M., & Singh, R. P. (2022). An era of ChatGPT as a significant futuristic support tool: A study on features, abilities, and challenges. *BenchCouncil Transactions on Benchmarks, Standards and Evaluations*, *2*(4), 100089. https://doi.org/10.1016/j.tbench.2023.100089

Härlin, T., Björnsson Rova, G., Sokolov, O., Singla, A., & Sukharevsky, A. (2023, April 26). *Exploring opportunities in the gen AI value chain | McKinsey*. QuantumBlack AI by McKinsey. https://www.mckinsey.com/capabilities/quantumblack/our-insights/exploring-opportunities-in-the-generative-ai-value-chain?cid=other--soc-lkn-bam-ip-mip----&sid=soc-POST_ID&linkId=221867230

Heinrich, L. J., & Häntschel, I. (2018). *Evaluation und Evaluationsforschung in der Wirtschaftsinformatik: Handbuch für Praxis, Lehre und Forschung* (Reprint 2018 ed. edition (Dec 16 1999)). Walter de Gruyter GmbH & Co KG.

Heinrich, L. J., Riedl, R., & Heinzl, A. (2011). Wirtschaftsinformatik Einführung und Grundlagen. In L. J. Heinrich, A. Heinzl, & R. Riedl (Hrsg.), *Wirtschaftsinformatik: Einführung und Grundlegung* (S. 319–330). Springer. https://doi.org/10.1007/978-3-642-15426-3_26

Hughes, A. (2023, Mai 30). *ChatGPT: Everything you need to know about OpenAI's GPT-4 tool*. BBC Science Focus Magazine. https://www.sciencefocus.com/future-technology/gpt-3/

Johner, P. D. C. (2015, August 10). ISO 25010 – ISO 9126 ~ Norm für Qualitätsmerkmale (Software). *Wissen zu medizinischer Software*. https://www.johner-institut.de/blog/iec-62304-medizinische-software/iso-9126-und-iso-25010/

Kanellopoulos, Y., Antonellis, P., Antoniou, D., Makris, C., Theodoridis, E., Tjortjis, C., & Tsirakis, N. (2010). Code Quality Evaluation Methodology Using The ISO/IEC 9126 Standard. *International Journal of Software Engineering & Applications*, *1*(3), 17–36. https://doi.org/10.5121/ijsea.2010.1302

Karampatsis, R.-M., & Sutton, C. (2019). *Maybe Deep Neural Networks are the Best Choice for Modeling Source Code* (arXiv:1903.05734). arXiv. http://arxiv.org/abs/1903.05734

Klima, M., Bures, M., Frajtak, K., Rechtberger, V., Trnka, M., Bellekens, X., Cerny, T., & Ahmed, B. S. (2022). Selected Code-Quality Characteristics and Metrics for Internet of Things Systems. *IEEE Access*, *10*, 46144–46161. https://doi.org/10.1109/ACCESS.2022.3170475

Kontogiannis, K., Martin, J., Wong, K., Gregory, R., Müller, H., & Mylopoulos, J. (2010). Code migration through transformations: An experience report. *CASCON First Decade High Impact Papers*, 201–213. https://doi.org/10.1145/1925805.1925817

Kurpicz-Briki, M. (2023). Generative AI: Was ist das und was kann sie bereits?

[Application/pdf]. *SocietyByte - Wissenschaftsmagazin der Berner Fachhochschule*. https://doi.org/10.24451/ARBOR.19044

Lachaux, M.-A., Roziere, B., Chanussot, L., & Lample, G. (2020). *Unsupervised Translation of Programming Languages* (arXiv:2006.03511). arXiv. https://doi.org/10.48550/arXiv.2006.03511

Lindrea, B. (2023, März 24). *ChatGPT can now access the internet with new OpenAI plugins*. Cointelegraph. https://cointelegraph.com/news/chatgpt-can-now-access-the-internet-with-new-openai-plugins

Liu, Y., Han, T., Ma, S., Zhang, J., Yang, Y., Tian, J., He, H., Li, A., Liu, Z., Wu, Z., Zhu, D., Li, X., Shen, D., & Liu, T. (2023). *Summary of ChatGPT/GPT-4 Research and Perspective Towards the Future of Large Language Models*.

Megahed, F. M., Chen, Y.-J., Ferris, J. A., Knoth, S., & Jones-Farmer, L. A. (2023). *How Generative AI models such as ChatGPT can be (Mis)Used in SPC Practice, Education, and Research? An Exploratory Study* (arXiv:2302.10916). arXiv. http://arxiv.org/abs/2302.10916

Meirelles, P., Santos Jr., C., Miranda, J., Kon, F., Terceiro, A., & Chavez, C. (2010). A Study of the Relationships between Source Code Metrics and Attractiveness in Free Software Projects. *2010 Brazilian Symposium on Software Engineering*, 11–20. https://doi.org/10.1109/SBES.2010.27

Mittal, S., & Bhatia, P. K. (2013). *Software Component Quality Models from ISO 9126 Perspective: A Review*. *02*(02).

Moradi Dakhel, A., Majdinasab, V., Nikanjam, A., Khomh, F., Desmarais, M. C., Ming, Z., & Jiang. (2022). GitHub Copilot AI pair programmer: Asset or Liability? In *ArXiv e-prints*. https://doi.org/10.48550/arXiv.2206.15331

Nguyen, N., & Nadi, S. (2022). An empirical evaluation of GitHub copilot's code suggestions. *Proceedings of the 19th International Conference on Mining Software Repositories*, 1–5. https://doi.org/10.1145/3524842.3528470

O'Hara, S. A. (2018). Improving Programming Language Transformation. *Proceedings of the International Conference on Software Engineering Research and Practice (SERP). The Steering Committee of The World Congress in Computer Science, Computer*, 129–135.

OpenAI. (o. J.). *GPT-4 is OpenAI's most advanced system, producing safer and more useful responses*. Abgerufen 3. Juli 2023, von https://openai.com/gpt-4

OpenAI. (2023). *GPT-4 Technical Report* (arXiv:2303.08774). arXiv. http://arxiv.org/abs/2303.08774

OpenAI. (2022a, Januar 27). *Aligning language models to follow instructions*. https://openai.com/research/instruction-following

OpenAI. (2022b, November 30). *Introducing ChatGPT*. https://openai.com/blog/chatgpt#OpenAI

Pearce, H., Ahmad, B., Tan, B., Dolan-Gavitt, B., & Karri, R. (2021). *Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions* (arXiv:2108.09293). arXiv. http://arxiv.org/abs/2108.09293

Pérez, F., Granger, B. E., & Hunter, J. D. (2011). Python: An Ecosystem for Scientific Computing. *Computing in Science & Engineering*, *13*(2), 13–21. https://doi.org/10.1109/MCSE.2010.119

Pudari, R. (2022). *AI Supported Software Development: Moving Beyond Code Completion* [Thesis]. https://dspace.library.uvic.ca/handle/1828/14155

Rahaman, M. S., Ahsan, M. M. T., Anjum, N., Terano, H. J. R., & Rahman, M. M. (2023). From ChatGPT-3 to GPT-4: A Significant Advancement in AI-Driven NLP Tools. *Journal of Engineering and Emerging Technologies*, *2*(1), 1–11. https://doi.org/10.52631/jeet.v2i1.188

Rentrop, J. (2006, August 31). *Software Metrics as Benchmarks for Source Code Quality of Software Systems*. https://homepages.cwi.nl/~paulk/thesesMasterSoftwareEngineering/2006/JulienRentrop.pdf

Rosenberg, D. L. H., & Hyatt, L. E. (1997). *Software Quality Metrics for Object-Oriented Environments*.

Rossum, G. van, & Drake, F. L. (2006). *An introduction to Python: Release 2.5* (2. print). Network Theory Limited.

Sammet, J. E. (1962). Basic elements of COBOL 61. *Communications of the ACM*, *5*(5), 237–253. https://doi.org/10.1145/367710.367721

Sharma, T., & Spinellis, D. (2020). *Do We Need Improved Code Quality Metrics?* (arXiv:2012.12324). arXiv. http://arxiv.org/abs/2012.12324

Sobania, D., Briesch, M., Hanna, C., & Petke, J. (2023). *An Analysis of the Automatic Bug Fixing Performance of ChatGPT* (arXiv:2301.08653). arXiv. http://arxiv.org/abs/2301.08653

Stack Overflow Developer Survey 2022. (2022, Mai). Stack Overflow. https://survey.stackoverflow.co/2022/?utm_source=social-share&utm_medium=social&utm_campaign=dev-survey-2022

Stamelos, I., Angelis, L., Oikonomou, A., & Bleris, G. L. (2002). Code quality analysis in open source software development. *Information Systems Journal*, *12*(1), 43–60. https://doi.org/10.1046/j.1365-2575.2002.00117.x

Steyer, R. (2018). Objektorientierte Programmierung in Python – Klassen, Objekte, Eigenschaften und Methoden. In R. Steyer (Hrsg.), *Programmierung in Python: Ein kompakter Einstieg für die Praxis* (S. 143–180). Springer Fachmedien. https://doi.org/10.1007/978-3-658-20705-2_11

Sun, J., Liao, Q. V., Muller, M., Agarwal, M., Houde, S., Talamadupula, K., & Weisz, J. D. (2022). Investigating Explainability of Generative AI for Code through Scenario-based Design. *27th International Conference on Intelligent User Interfaces*, 212–228. https://doi.org/10.1145/3490099.3511119

Surameery, N. M. S., & Shakor, M. Y. (2023). Use Chat GPT to Solve Programming Bugs. *International Journal of Information Technology & Computer Engineering (IJITC) ISSN : 2455-5290*, *3*(01), Article 01. https://doi.org/10.55529/ijitc.31.17.22

Szafraniec, M., Roziere, B., Leather, H., Charton, F., Labatut, P., & Synnaeve, G. (2023). *Code Translation with Compiler Representations* (arXiv:2207.03578). arXiv. http://arxiv.org/abs/2207.03578

Talamadupula, K. (2021). Applied AI matters: AI4Code: applying artificial intelligence to source code. *AI Matters*, *7*(1), 18–20. https://doi.org/10.1145/3465074.3465080

Tian, H., Lu, W., Li, T. O., Tang, X., Cheung, S.-C., Klein, J., & Bissyandé, T. F. (2023). *Is ChatGPT the Ultimate Programming Assistant—How far is it?*

Vaithilingam, P., Zhang, T., & Glassman, E. L. (2022). Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models. *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems*, 1–7. https://doi.org/10.1145/3491101.3519665

Vytovtov, P., & Markov, E. (2017). Source code quality classification based on software metrics. *2017 20th Conference of Open Innovations Association (FRUCT)*, 505–511. https://doi.org/10.23919/FRUCT.2017.8071355

Wang, R., Cheng, R., Ford, D., & Zimmermann, T. (2023). *Investigating and Designing for Trust in AI-powered Code Generation Tools* (arXiv:2305.11248). arXiv.

http://arxiv.org/abs/2305.11248

Weisz, J. D., Muller, M., He, J., & Houde, S. (2023). *Toward General Design Principles for Generative AI Applications* (arXiv:2301.05578). arXiv. http://arxiv.org/abs/2301.05578

Weisz, J. D., Muller, M., Houde, S., Richards, J., Ross, S. I., Martinez, F., Agarwal, M., & Talamadupula, K. (2021). Perfection Not Required? Human-AI Partnerships in Code Translation. *26th International Conference on Intelligent User Interfaces*, 402–412. https://doi.org/10.1145/3397481.3450656

Weisz, J. D., Muller, M., Ross, S. I., Martinez, F., Houde, S., Agarwal, M., Talamadupula, K., & Richards, J. T. (2022). Better Together? An Evaluation of AI-Supported Code Translation. *27th International Conference on Intelligent User Interfaces*, 369–391. https://doi.org/10.1145/3490099.3511157

Wong, D., Kothig, A., & Lam, P. (2022). *Exploring the Verifiability of Code Generated by GitHub Copilot* (arXiv:2209.01766). arXiv. https://doi.org/10.48550/arXiv.2209.01766

Zhou, J., Ke, P., Qiu, X., Huang, M., & Zhang, J. (2023). ChatGPT: Potential, prospects, and limitations. *Frontiers of Information Technology & Electronic Engineering*. https://doi.org/10.1631/FITEE.2300089

Ziegler, A., Kalliamvakou, E., Li, X. A., Rice, A., Rifkin, D., Simister, S., Sittampalam, G., & Aftandilian, E. (2022). Productivity assessment of neural code completion. *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*, 21–29. https://doi.org/10.1145/3520312.3534864

Zong, M., & Krishnamachari, B. (2022). *A survey on GPT-3* (arXiv:2212.00857). arXiv. https://doi.org/10.48550/arXiv.2212.00857

10 Anhang

10.1 Anhang A: Resultate Bewertung von ChatGPT als Codeübersetzungstool

Resultate Bewertung von ChatGPT als Codeübersetzungstool															
Bezeichnung Beispiel	Gleichheit der Ausgaben	Vollständigkeit der Übersetzung	Richtigkeit	**Funktionalität**	Grösse des Programms	Redundanz	Einhaltung von Best Practices	**Optimalität**	Lesbarkeit	Variablen und Funktionen	Kommentare	**Verständlichkeit**	**Gesamt**	**Prozentual**
addamount_unsup	2	2	2	**6**	2	1	1	**4**	2	1	0	**3**	**13**	**72%**
addamount_sup	2	2	2	**6**	2	1	1	**4**	2	1	2	**5**	**15**	**83%**
bubblesort_unsup	2	2	2	**6**	2	2	2	**6**	2	1	0	**3**	**15**	**83%**
bubblesort_sup	1	2	2	**5**	2	2	1	**5**	2	1	2	**5**	**15**	**83%**
flatprice_unsup	2	2	2	**6**	1	0	1	**2**	2	2	0	**4**	**12**	**67%**
flatprice_sup	2	2	2	**6**	1	1	1	**3**	1	2	2	**5**	**14**	**78%**
forloop_unsup	1	2	2	**5**	2	2	2	**6**	1	1	0	**2**	**13**	**72%**
forloop_sup	0	2	2	**4**	2	2	1	**5**	2	1	2	**5**	**14**	**78%**
helloworld2_unsup	2	2	2	**6**	2	2	2	**6**	2	2	2	**6**	**18**	**100%**
helloworld2_sup	2	2	2	**6**	2	2	2	**6**	2	2	2	**6**	**18**	**100%**
~~isnumeric_unsup~~														
~~isnumeric_sup~~														
jsongenerate_unsup	2	2	2	**6**	2	2	1	**5**	2	1	1	**4**	**15**	**83%**
jsongenerate_sup	2	2	2	**6**	1	2	1	**4**	2	1	2	**5**	**15**	**83%**
~~mergesort_unsup~~														
~~mergesort_sup~~														
notenrechner_unsup	0	0	2	**2**	0	0	0	**0**	0	0	0	**0**	**2**	**11%**
notenrechner_sup	2	2	1	**5**	0	0	0	**0**	0	0	0	**0**	**5**	**28%**
npkcalculator_unsup	2	2	2	**6**	2	2	1	**5**	2	2	0	**4**	**15**	**83%**
npkcalculator_sup	2	2	2	**6**	1	2	0	**3**	1	2	1	**4**	**13**	**72%**
numval_unsup	1	2	2	**5**	2	2	1	**5**	2	1	1	**4**	**14**	**78%**
numval_sup	1	2	2	**5**	2	2	2	**6**	2	1	2	**5**	**16**	**89%**
payroll_unsup	2	2	2	**6**	2	2	1	**5**	2	2	0	**4**	**15**	**83%**
payroll_sup	2	2	2	**6**	2	2	2	**6**	2	2	2	**6**	**18**	**100%**
randombingo_unsup	0	0	0	**0**	0	1	0	**1**	0	0	0	**0**	**1**	**6%**
randombingo_sup	2	2	2	**6**	0	1	0	**1**	0	1	1	**2**	**9**	**50%**
randomnumber_unsup	2	2	2	**6**	2	2	1	**5**	2	1	0	**3**	**14**	**78%**
randomnumber_sup	2	2	2	**6**	1	2	1	**4**	2	1	2	**5**	**15**	**83%**
randomsort_unsup	2	2	2	**6**	2	2	1	**5**	2	1	1	**4**	**15**	**83%**
randomsort_sup	2	2	2	**6**	2	2	2	**6**	2	1	2	**5**	**17**	**94%**
simplecalculator_unsup	2	2	2	**6**	2	2	1	**5**	2	1	0	**3**	**14**	**78%**
simplecalculator_sup	1	2	2	**5**	1	1	1	**3**	1	1	0	**2**	**10**	**56%**
stringsplit_unsup	2	2	2	**6**	2	2	1	**5**	2	1	0	**3**	**14**	**78%**
stringsplit_sup	2	2	2	**6**	1	1	1	**3**	1	1	2	**4**	**13**	**72%**
trim_unsup	2	2	2	**6**	0	0	0	**0**	0	0	0	**0**	**6**	**33%**
trim_sup	2	2	2	**6**	0	0	0	**0**	0	0	1	**1**	**7**	**39%**
~~unstring_unsup~~														
~~unstring_sup~~														
whileloop_unsup	1	2	2	**5**	2	2	2	**6**	2	1	0	**3**	**14**	**78%**
whileloop_sup	1	2	2	**5**	2	2	2	**6**	2	1	0	**3**	**14**	**78%**
Durchschnittswerte ohne Anweisung	**1.588235294**	**1.764705882**	**1.882352941**	**5.235294118**	**1.588235294**	**1.529411765**	**1.058823529**	**4.176470588**	**1.588235294**	**1.058823529**	**0.294117647**	**2.941176471**	**12.35294118**	**69%**
Durchschnittswerte mit Anweisung	**1.647058824**	**2**	**1.941176471**	**5.588235294**	**1.294117647**	**1.470588235**	**1.058823529**	**3.823529412**	**1.411764706**	**1.117647059**	**1.470588235**	**4**	**13.41176471**	**75%**
Durschnittswerte gesamt	**1.617647059**	**1.882352941**	**1.911764706**	**5.411764706**	**1.441176471**	**1.5**	**1.058823529**	**4**	**1.5**	**1.088235294**	**0.882352941**	**3.470588235**	**12.88235294**	**72%**

10.2 Anhang B: Bewertungsraster pro Codebeispiel

Addamount

Kriterien zur Bewertung von ChatGPT (ohne Anweisung)					
Punkte	**Funktionalität**				
	Gleichheit der Ausgaben	**Vollständigkeit der Übersetzung**	**Richtigkeit**		
2	Die Ausgaben des Python-Codes und des COBOL-Codes sind identisch.	Der Python-Code übersetzt und implementiert alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert ohne Fehler und alle Funktionen können ohne Ausführungsfehler ausgeführt werden.		
1	Die Ausgaben des Python-Codes und des COBOL-Codes sind ähnlich, aber nicht identisch.	Der Python-Code übersetzt und implementiert die meisten, aber nicht alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert mit einigen Warnungen, oder es treten bei der Ausführung einige kleinere Fehler auf, die jedoch die grundsätzliche Funktionsfähigkeit nicht beeinträchtigen. Fehler lassen sich mit wenigen zusätzlichen Prompts beheben.		
0	Die Ausgaben des Python-Codes und des COBOL-Codes unterscheiden sich erheblich.	Der Python-Code lässt wesentliche Funktionen, Verfahren oder Operationen des ursprünglichen COBOL-Codes aus oder implementiert sie falsch.	Der Python-Code kompiliert nicht, oder es treten bei der Ausführung schwere Fehler auf, die die Funktionsfähigkeit beeinträchtigen. Fehler lassen sich kaum beheben.		
				Total Punkte	6
				Erreichte Punkte	6
				Erreichte Punkte in Prozent	100
	Optimalität				
	Grösse des Programms	**Redundanz**	**Einhaltung von Best Practices**		
2	Der Python-Code ist effizient geschrieben und so gut wie möglich komprimiert. Es gibt eine angemessene Anzahl von Leerzeilen zur Strukturierung.	Der Python-Code vermeidet redundante Operationen oder Wiederholungen. Code-Wiederverwendung wird maximiert.	Der Python-Code hält sich an allgemein anerkannte Best Practices für Python-Programmierung, einschliesslich Einrückung, Benennungskonventionen, Verwendung von Funktionen und Klassen usw.		
1	Der Python-Code ist im Allgemeinen effizient geschrieben, es gibt jedoch Möglichkeiten zur Optimierung.	Der Python-Code enthält einige redundante Operationen oder Wiederholungen, die reduziert werden könnten.	Der Python-Code hält sich grösstenteils an Best Practices, es gibt jedoch einige Verstösse.		
0	Der Python-Code ist ineffizient und zu lang. Es gibt zu viele Leerzeilen.	Der Python-Code ist voller redundanter Operationen oder Wiederholungen.	Der Python-Code verstösst häufig gegen Best Practices.		
				Total Punkte	6
				Erreichte Punkte	4
				Erreichte Punkte in Prozent	66.66667
	Verständlichkeit				
	Lesbarkeit	**Variablen und Funktionen**	**Kommentare**		
2	Der Python-Code ist einfach zu lesen und zu	Variablen und Funktionen haben klare,	Der Code ist gut kommentiert, und die		
1	Der Python-Code ist verständlich, enthält aber unklare Abschnitte.	Die meisten Variablen und Funktionen haben klare, beschreibende Namen, aber einige könnten verbessert werden.	Der Code hat einige nützliche Kommentare, aber es gibt Bereiche, die mehr Kommentierung erfordern oder wo die vorhandenen Kommentare nicht klar sind.		
0	Der Python-Code ist schwer zu lesen und zu verstehen.	Variablen und Funktionen haben unklare oder nicht beschreibende Namen.	Der Code hat wenige oder keine hilfreichen Kommentare.		
				Total Punkte	6
				Erreichte Punkte	3
				Erreichte Punkte in Prozent	50

Kommentar:
- COBOL Output füllt INT links mit Nullen auf, Python Code füllt nicht mit Nullen auf
- Python Code ist verbessert, da COBOL Code negativen Input nicht als negativ interpretiert. Übersetzung wählt besser geeigneten Datentyp
- Zeilen 8 - 10 können in For-Loop abgebildet werden. So wie der Code geschrieben ist, müsste für jeden weiteren Input eine neue Zeile hinzugefügt werden
- Klasse wird verwendet obwohl nicht nötig, Programm hätte auf main-Funktion reduziert werden können. So hätte man sich Code sparen können
- Variablennamen übernommen. Funktion nicht zielführend benannt -> Main verwendet anstatt verständlicher Name

Kriterien zur Bewertung von ChatGPT (mit Anweisung)					
Punkte	**Funktionalität**				
	Gleichheit der Ausgaben	**Vollständigkeit der Übersetzung**	**Richtigkeit**		
2	Die Ausgaben des Python-Codes und des COBOL-Codes sind identisch.	Der Python-Code übersetzt und implementiert alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert ohne Fehler und alle Funktionen können ohne Ausführungsfehler ausgeführt werden.		
1	Die Ausgaben des Python-Codes und des COBOL-Codes sind ähnlich, aber nicht identisch.	Der Python-Code übersetzt und implementiert die meisten, aber nicht alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert mit einigen Warnungen, oder es treten bei der Ausführung einige kleinere Fehler auf, die jedoch die grundsätzliche Funktionsfähigkeit nicht beeinträchtigen. Fehler lassen sich mit wenigen zusätzlichen Prompts beheben.		
0	Die Ausgaben des Python-Codes und des COBOL-Codes unterscheiden sich erheblich.	Der Python-Code lässt wesentliche Funktionen, Verfahren oder Operationen des ursprünglichen COBOL-Codes aus oder implementiert sie falsch.	Der Python-Code kompiliert nicht, oder es treten bei der Ausführung schwere Fehler auf, die die Funktionsfähigkeit beeinträchtigen. Fehler lassen sich kaum beheben		
				Total Punkte	6
				Erreichte Punkte	6
				Erreichte Punkte in Prozent	100
	Optimalität				
	Grösse des Programms	**Redundanz**	**Einhaltung von Best Practices**		
2	Der Python-Code ist effizient geschrieben und so gut wie möglich komprimiert. Es gibt eine angemessene Anzahl von Leerzeilen zur Strukturierung.	Der Python-Code vermeidet redundante Operationen oder Wiederholungen. Code-Wiederverwendung wird maximiert.	Der Python-Code hält sich an allgemein anerkannte Best Practices für Python-Programmierung, einschliesslich Einrückung, Benennungskonventionen, Verwendung von Funktionen und Klassen usw.		
1	Der Python-Code ist im Allgemeinen effizient geschrieben, es gibt jedoch Möglichkeiten zur Optimierung.	Der Python-Code enthält einige redundante Operationen oder Wiederholungen, die reduziert werden könnten.	Der Python-Code hält sich grösstenteils an Best Practices, es gibt jedoch einige Verstösse.		
0	Der Python-Code ist ineffizient und zu lang. Es gibt zu viele Leerzeilen.	Der Python-Code ist voller redundanter Operationen oder Wiederholungen.	Der Python-Code verstösst häufig gegen Best Practices.		
				Total Punkte	6
				Erreichte Punkte	4
				Erreichte Punkte in Prozent	66.66667
	Verständlichkeit				
	Lesbarkeit	**Variablen und Funktionen**	**Kommentare**		
2	Der Python-Code ist einfach zu lesen und zu	Variablen und Funktionen haben klare,	Der Code ist gut kommentiert, und die		
1	Der Python-Code ist verständlich, enthält aber unklare Abschnitte.	Die meisten Variablen und Funktionen haben klare, beschreibende Namen, aber einige könnten verbessert werden.	Der Code hat einige nützliche Kommentare, aber es gibt Bereiche, die mehr Kommentierung erfordern oder wo die vorhandenen Kommentare nicht klar sind.		
0	Der Python-Code ist schwer zu lesen und zu verstehen.	Variablen und Funktionen haben unklare oder nicht beschreibende Namen.	Der Code hat wenige oder keine hilfreichen Kommentare.		
				Total Punkte	6
				Erreichte Punkte	5
				Erreichte Punkte in Prozent	83.33333
Kommentar:					
- COBOL Output füllt INT links mit Nullen auf, Python Code füllt nicht mit Nullen auf					
- Python Code ist verbessert, da COBOL Code negativen Input nicht als negativ interpretiert. Übersetzung wählt besser geeigneten Datentyp					
- Zeilen 5-7 können in For-Loop abgebildet werden. So wie der Code geschrieben ist, müsste für jeden weiteren Input eine neue Zeile hinzugefügt werden					
- Enthält keine Main-Methode, kann zu Problemen führen wenn Script importiert und in anderem Code ausgeführt wird.					
- Zeile 10 führt sinnlose Variable ein, hätte weiter cust_no_in verwenden können. Keine Funktion wird verwendet					

Bubblesort

Kriterien zur Bewertung von ChatGPT (ohne Anweisung)					
Punkte	**Funktionalität**				
	Gleichheit der Ausgaben	**Vollständigkeit der Übersetzung**	**Richtigkeit**		
2	Die Ausgaben des Python-Codes und des COBOL-Codes sind identisch.	Der Python-Code übersetzt und implementiert alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert ohne Fehler und alle Funktionen können ohne Ausführungsfehler ausgeführt werden.		
1	Die Ausgaben des Python-Codes und des COBOL-Codes sind ähnlich, aber nicht identisch.	Der Python-Code übersetzt und implementiert die meisten, aber nicht alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert mit einigen Warnungen, oder es treten bei der Ausführung einige kleinere Fehler auf, die jedoch die grundsätzliche Funktionsfähigkeit nicht beeinträchtigen. Fehler lassen sich mit wenigen zusätzlichen Prompts beheben.		
0	Die Ausgaben des Python-Codes und des COBOL-Codes unterscheiden sich erheblich.	Der Python-Code lässt wesentliche Funktionen, Verfahren oder Operationen des ursprünglichen COBOL-Codes aus oder implementiert sie falsch.	Der Python-Code kompiliert nicht, oder es treten bei der Ausführung schwere Fehler auf, die die Funktionsfähigkeit beeinträchtigen. Fehler lassen sich kaum beheben		
				Total Punkte	6
				Erreichte Punkte	6
				Erreichte Punkte in Prozent	100
	Optimalität				
	Grösse des Programms	**Redundanz**	**Einhaltung von Best Practices**		
2	Der Python-Code ist effizient geschrieben und so gut wie möglich komprimiert. Es gibt eine angemessene Anzahl von Leerzeilen zur Strukturierung.	Der Python-Code vermeidet redundante Operationen oder Wiederholungen. Code-Wiederverwendung wird maximiert.	Der Python-Code hält sich an allgemein anerkannte Best Practices für Python-Programmierung, einschliesslich Einrückung, Benennungskonventionen, Verwendung von Funktionen und Klassen usw.		
1	Der Python-Code ist im Allgemeinen effizient geschrieben, es gibt jedoch Möglichkeiten zur Optimierung.	Der Python-Code enthält einige redundante Operationen oder Wiederholungen, die reduziert werden könnten.	Der Python-Code hält sich grösstenteils an Best Practices, es gibt jedoch einige Verstösse.		
0	Der Python-Code ist ineffizient und zu lang. Es gibt zu viele Leerzeilen.	Der Python-Code ist voller redundanter Operationen oder Wiederholungen.	Der Python-Code verstösst häufig gegen Best Practices.		
				Total Punkte	6
				Erreichte Punkte	6
				Erreichte Punkte in Prozent	100
	Verständlichkeit				
	Lesbarkeit	**Variablen und Funktionen**	**Kommentare**		
2	Der Python-Code ist einfach zu lesen und zu verstehen.	Variablen und Funktionen haben klare, beschreibende Namen, die ihrem Zweck im Code entsprechen.	Der Code ist gut kommentiert, und die Kommentare tragen effektiv zum Verständnis des Codes bei.		
1	Der Python-Code ist verständlich, enthält aber unklare Abschnitte.	Die meisten Variablen und Funktionen haben klare, beschreibende Namen, aber einige könnten verbessert werden.	Der Code hat einige nützliche Kommentare, aber es gibt Bereiche, die mehr Kommentierung erfordern oder wo die vorhandenen Kommentare nicht klar sind.		
0	Der Python-Code ist schwer zu lesen und zu verstehen.	Variablen und Funktionen haben unklare oder nicht beschreibende Namen.	Der Code hat wenige oder keine hilfreichen Kommentare.		
				Total Punkte	6
				Erreichte Punkte	3
				Erreichte Punkte in Prozent	50
Kommentar:					
- Importiert Modul random selbstständig					
- Code macht so wie angewandt Sinn, auch Klassenbezeichnung. Funtkion generate_random_num sorting_array hätte besser benannt werden können					

Kriterien zur Bewertung von ChatGPT (mit Anweisung)					
Punkte	**Funktionalität**				
	Gleichheit der Ausgaben	**Vollständigkeit der Übersetzung**	**Richtigkeit**		
2	Die Ausgaben des Python-Codes und des COBOL-Codes sind identisch.	Der Python-Code übersetzt und implementiert alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert ohne Fehler und alle Funktionen können ohne Ausführungsfehler ausgeführt werden.		
1	Die Ausgaben des Python-Codes und des COBOL-Codes sind ähnlich, aber nicht identisch.	Der Python-Code übersetzt und implementiert die meisten, aber nicht alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert mit einigen Warnungen, oder es treten bei der Ausführung einige kleinere Fehler auf, die jedoch die grundsätzliche Funktionsfähigkeit nicht beeinträchtigen. Fehler lassen sich mit wenigen zusätzlichen Prompts beheben.		
0	Die Ausgaben des Python-Codes und des COBOL-Codes unterscheiden sich erheblich.	Der Python-Code lässt wesentliche Funktionen, Verfahren oder Operationen des ursprünglichen COBOL-Codes aus oder implementiert sie falsch.	Der Python-Code kompiliert nicht, oder es treten bei der Ausführung schwere Fehler auf, die die Funktionsfähigkeit beeinträchtigen. Fehler lassen sich kaum beheben		
				Total Punkte	6
				Erreichte Punkte	5
				Erreichte Punkte in Prozent	83.33333
	Optimalität				
	Grösse des Programms	**Redundanz**	**Einhaltung von Best Practices**		
2	Der Python-Code ist effizient geschrieben und so gut wie möglich komprimiert. Es gibt eine angemessene Anzahl von Leerzeilen zur Strukturierung.	Der Python-Code vermeidet redundante Operationen oder Wiederholungen. Code-Wiederverwendung wird maximiert.	Der Python-Code hält sich an allgemein anerkannte Best Practices für Python-Programmierung, einschliesslich Einrückung, Benennungskonventionen, Verwendung von Funktionen und Klassen usw.		
1	Der Python-Code ist im Allgemeinen effizient geschrieben, es gibt jedoch Möglichkeiten zur Optimierung.	Der Python-Code enthält einige redundante Operationen oder Wiederholungen, die reduziert werden könnten.	Der Python-Code hält sich grösstenteils an Best Practices, es gibt jedoch einige Verstösse.		
0	Der Python-Code ist ineffizient und zu lang. Es gibt zu viele Leerzeilen.	Der Python-Code ist voller redundanter Operationen oder Wiederholungen.	Der Python-Code verstösst häufig gegen Best Practices.		
				Total Punkte	6
				Erreichte Punkte	5
				Erreichte Punkte in Prozent	83.33333
	Verständlichkeit				
	Lesbarkeit	**Variablen und Funktionen**	**Kommentare**		
2	Der Python-Code ist einfach zu lesen und zu verstehen.	Variablen und Funktionen haben klare, beschreibende Namen, die ihrem Zweck im Code entsprechen.	Der Code ist gut kommentiert, und die Kommentare tragen effektiv zum Verständnis des Codes bei.		
1	Der Python-Code ist verständlich, enthält aber unklare Abschnitte.	Die meisten Variablen und Funktionen haben klare, beschreibende Namen, aber einige könnten verbessert werden.	Der Code hat einige nützliche Kommentare, aber es gibt Bereiche, die mehr Kommentierung erfordern oder wo die vorhandenen Kommentare nicht klar sind.		
0	Der Python-Code ist schwer zu lesen und zu verstehen.	Variablen und Funktionen haben unklare oder nicht beschreibende Namen.	Der Code hat wenige oder keine hilfreichen Kommentare.		
				Total Punkte	6
				Erreichte Punkte	5
				Erreichte Punkte in Prozent	83.33333
Kommentar:					
- Verwendet keine Main Methode					
- Variablennamen könnten aussagekräftiger sein					
- Output nicht identisch, in diesem Code ist der Output in einer Liste dargestellt.					

Flatprice

Kriterien zur Bewertung von ChatGPT (ohne Anweisung)					
Punkte	**Funktionalität**				
	Gleichheit der Ausgaben	**Vollständigkeit der Übersetzung**	**Richtigkeit**		
2	Die Ausgaben des Python-Codes und des COBOL-Codes sind identisch.	Der Python-Code übersetzt und implementiert alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert ohne Fehler und alle Funktionen können ohne Ausführungsfehler ausgeführt werden.		
1	Die Ausgaben des Python-Codes und des COBOL-Codes sind ähnlich, aber nicht identisch.	Der Python-Code übersetzt und implementiert die meisten, aber nicht alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert mit einigen Warnungen, oder es treten bei der Ausführung einige kleinere Fehler auf, die jedoch die grundsätzliche Funktionsfähigkeit nicht beeinträchtigen. Fehler lassen sich mit wenigen zusätzlichen Prompts beheben.		
0	Die Ausgaben des Python-Codes und des COBOL-Codes unterscheiden sich erheblich.	Der Python-Code lässt wesentliche Funktionen, Verfahren oder Operationen des ursprünglichen COBOL-Codes aus oder implementiert sie falsch.	Der Python-Code kompiliert nicht, oder es treten bei der Ausführung schwere Fehler auf, die die Funktionsfähigkeit beeinträchtigen. Fehler lassen sich kaum beheben		
				Total Punkte	6
				Erreichte Punkte	6
				Erreichte Punkte in Prozent	100
	Optimalität				
	Grösse des Programms	**Redundanz**	**Einhaltung von Best Practices**		
2	Der Python-Code ist effizient geschrieben und so gut wie möglich komprimiert. Es gibt eine angemessene Anzahl von Leerzeilen zur Strukturierung.	Der Python-Code vermeidet redundante Operationen oder Wiederholungen. Code-Wiederverwendung wird maximiert.	Der Python-Code hält sich an allgemein anerkannte Best Practices für Python-Programmierung, einschliesslich Einrückung, Benennungskonventionen, Verwendung von Funktionen und Klassen usw.		
1	Der Python-Code ist im Allgemeinen effizient geschrieben, es gibt jedoch Möglichkeiten zur Optimierung.	Der Python-Code enthält einige redundante Operationen oder Wiederholungen, die reduziert werden könnten.	Der Python-Code hält sich grösstenteils an Best Practices, es gibt jedoch einige Verstösse.		
0	Der Python-Code ist ineffizient und zu lang. Es gibt zu viele Leerzeilen.	Der Python-Code ist voller redundanter Operationen oder Wiederholungen.	Der Python-Code verstösst häufig gegen Best Practices.		
				Total	6
				Erreichte Punkte	2
				Erreichte Punkte in Prozent	33.33333
	Verständlichkeit				
	Lesbarkeit	**Variablen und Funktionen**	**Kommentare**		
2	Der Python-Code ist einfach zu lesen und zu verstehen.	Variablen und Funktionen haben klare, beschreibende Namen, die ihrem Zweck im Code entsprechen.	Der Code ist gut kommentiert, und die Kommentare tragen effektiv zum Verständnis des Codes bei.		
1	Der Python-Code ist verständlich, enthält aber unklare Abschnitte.	Die meisten Variablen und Funktionen haben klare, beschreibende Namen, aber einige könnten verbessert werden.	Der Code hat einige nützliche Kommentare, aber es gibt Bereiche, die mehr Kommentierung erfordern oder wo die vorhandenen Kommentare nicht klar sind.		
0	Der Python-Code ist schwer zu lesen und zu verstehen.	Variablen und Funktionen haben unklare oder nicht beschreibende Namen.	Der Code hat wenige oder keine hilfreichen Kommentare.		
				Total Punkte	6
				Erreichte Punkte	4
				Erreichte Punkte in Prozent	66.66667
Kommentar: - übersetzt das Programm 1:1, auch wenn das nicht effizient ist - Programm könnte effizienter geschrieben sein, z.B. print outputs in Schleife packen - Verwendet Main Methode - Variablen werden aus COBOL Programm übernommen					

Kriterien zur Bewertung von ChatGPT (mit Anweisung)					
Punkte	**Funktionalität**				
	Gleichheit der Ausgaben	**Vollständigkeit der Übersetzung**	**Richtigkeit**		
2	Die Ausgaben des Python-Codes und des COBOL-Codes sind identisch.	Der Python-Code übersetzt und implementiert alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert ohne Fehler und alle Funktionen können ohne Ausführungsfehler ausgeführt werden.		
1	Die Ausgaben des Python-Codes und des COBOL-Codes sind ähnlich, aber nicht identisch.	Der Python-Code übersetzt und implementiert die meisten, aber nicht alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert mit einigen Warnungen, oder es treten bei der Ausführung einige kleinere Fehler auf, die jedoch die grundsätzliche Funktionsfähigkeit nicht beeinträchtigen. Fehler lassen sich mit wenigen zusätzlichen Prompts beheben.		
0	Die Ausgaben des Python-Codes und des COBOL-Codes unterscheiden sich erheblich.	Der Python-Code lässt wesentliche Funktionen, Verfahren oder Operationen des ursprünglichen COBOL-Codes aus oder implementiert sie falsch.	Der Python-Code kompiliert nicht, oder es treten bei der Ausführung schwere Fehler auf, die die Funktionsfähigkeit beeinträchtigen. Fehler lassen sich kaum beheben		
				Total Punkte	6
				Erreichte Punkte	6
				Erreichte Punkte in Prozent	100
	Optimalität				
	Grösse des Programms	**Redundanz**	**Einhaltung von Best Practices**		
2	Der Python-Code ist effizient geschrieben und so gut wie möglich komprimiert. Es gibt eine angemessene Anzahl von Leerzeilen zur Strukturierung.	Der Python-Code vermeidet redundante Operationen oder Wiederholungen. Code-Wiederverwendung wird maximiert.	Der Python-Code hält sich an allgemein anerkannte Best Practices für Python-Programmierung, einschliesslich Einrückung, Benennungskonventionen, Verwendung von Funktionen und Klassen usw.		
1	Der Python-Code ist im Allgemeinen effizient geschrieben, es gibt jedoch Möglichkeiten zur Optimierung.	Der Python-Code enthält einige redundante Operationen oder Wiederholungen, die reduziert werden könnten.	Der Python-Code hält sich grösstenteils an Best Practices, es gibt jedoch einige Verstösse.		
0	Der Python-Code ist ineffizient und zu lang. Es gibt zu viele Leerzeilen.	Der Python-Code ist voller redundanter Operationen oder Wiederholungen.	Der Python-Code verstösst häufig gegen Best Practices.		
				Total	6
				Erreichte Punkte	3
				Erreichte Punkte in Prozent	50
	Verständlichkeit				
	Lesbarkeit	**Variablen und Funktionen**	**Kommentare**		
2	Der Python-Code ist einfach zu lesen und zu verstehen.	Variablen und Funktionen haben klare, beschreibende Namen, die ihrem Zweck im Code entsprechen.	Der Code ist gut kommentiert, und die Kommentare tragen effektiv zum Verständnis des Codes bei.		
1	Der Python-Code ist verständlich, enthält aber unklare Abschnitte.	Die meisten Variablen und Funktionen haben klare, beschreibende Namen, aber einige könnten verbessert werden.	Der Code hat einige nützliche Kommentare, aber es gibt Bereiche, die mehr Kommentierung erfordern oder wo die vorhandenen Kommentare nicht klar sind.		
0	Der Python-Code ist schwer zu lesen und zu verstehen.	Variablen und Funktionen haben unklare oder nicht beschreibende Namen.	Der Code hat wenige oder keine hilfreichen Kommentare.		
				Total Punkte	6
				Erreichte Punkte	5
				Erreichte Punkte in Prozent	83.33333
Kommentare					
- Verwendet keine Main Methode					
- Führt ohne Sinn neue Variablen ein, indem ein identischer Wert gespeichert wird, und gibt dann jeweils 2 mal den selben Output aus					
- Code dadurch deutlich länger als nötig					

Forloop

Kriterien zur Bewertung von ChatGPT (ohne Anweisung)					
Punkte	**Funktionalität**				
	Gleichheit der Ausgaben	**Vollständigkeit der Übersetzung**	**Richtigkeit**		
2	Die Ausgaben des Python-Codes und des COBOL-Codes sind identisch.	Der Python-Code übersetzt und implementiert alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert ohne Fehler und alle Funktionen können ohne Ausführungsfehler ausgeführt werden.		
1	Die Ausgaben des Python-Codes und des COBOL-Codes sind ähnlich, aber nicht identisch.	Der Python-Code übersetzt und implementiert die meisten, aber nicht alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert mit einigen Warnungen, oder es treten bei der Ausführung einige kleinere Fehler auf, die jedoch die grundsätzliche Funktionsfähigkeit nicht beeinträchtigen. Fehler lassen sich mit wenigen zusätzlichen Prompts beheben.		
0	Die Ausgaben des Python-Codes und des COBOL-Codes unterscheiden sich erheblich.	Der Python-Code lässt wesentliche Funktionen, Verfahren oder Operationen des ursprünglichen COBOL-Codes aus oder implementiert sie falsch.	Der Python-Code kompiliert nicht, oder es treten bei der Ausführung schwere Fehler auf, die die Funktionsfähigkeit beeinträchtigen. Fehler lassen sich kaum beheben		
				Total Punkte	6
				Erreichte Punkte	5
				Erreichte Punkte in Prozent	83.33333
	Optimalität				
	Grösse des Programms	**Redundanz**	**Einhaltung von Best Practices**		
2	Der Python-Code ist effizient geschrieben und so gut wie möglich komprimiert. Es gibt eine angemessene Anzahl von Leerzeilen zur Strukturierung.	Der Python-Code vermeidet redundante Operationen oder Wiederholungen. Code-Wiederverwendung wird maximiert.	Der Python-Code hält sich an allgemein anerkannte Best Practices für Python-Programmierung, einschliesslich Einrückung, Benennungskonventionen, Verwendung von Funktionen und Klassen usw.		
1	Der Python-Code ist im Allgemeinen effizient geschrieben, es gibt jedoch Möglichkeiten zur Optimierung.	Der Python-Code enthält einige redundante Operationen oder Wiederholungen, die reduziert werden könnten.	Der Python-Code hält sich grösstenteils an Best Practices, es gibt jedoch einige Verstösse.		
0	Der Python-Code ist ineffizient und zu lang. Es gibt zu viele Leerzeilen.	Der Python-Code ist voller redundanter Operationen oder Wiederholungen.	Der Python-Code verstösst häufig gegen Best Practices.		
				Total Punkte	6
				Erreichte Punkte	6
				Erreichte Punkte in Prozent	100
	Verständlichkeit				
	Lesbarkeit	**Variablen und Funktionen**	**Kommentare**		
2	Der Python-Code ist einfach zu lesen und zu	Variablen und Funktionen haben klare,	Der Code ist gut kommentiert, und die		
1	Der Python-Code ist verständlich, enthält aber unklare Abschnitte.	Die meisten Variablen und Funktionen haben klare, beschreibende Namen, aber einige könnten verbessert werden.	Der Code hat einige nützliche Kommentare, aber es gibt Bereiche, die mehr Kommentierung erfordern oder wo die vorhandenen Kommentare nicht klar sind.		
0	Der Python-Code ist schwer zu lesen und zu verstehen.	Variablen und Funktionen haben unklare oder nicht beschreibende Namen.	Der Code hat wenige oder keine hilfreichen Kommentare.		
				Total Punkte	6
				Erreichte Punkte	2
				Erreichte Punkte in Prozent	33.33333

Kommentar:
- Macht dasselbe
- Variablen und Funktionsnamen wurden 1:1 übernommen, machen den Code aber nicht sehr leserlich
- Enthält keine Kommentare
- Ausgaben sind nicht mit der gleichen Logik sortiert

Kriterien zur Bewertung von ChatGPT (mit Anweisung)					
Punkte	**Funktionalität**				
	Gleichheit der Ausgaben	**Vollständigkeit der Übersetzung**	**Richtigkeit**		
2	Die Ausgaben des Python-Codes und des COBOL-Codes sind identisch.	Der Python-Code übersetzt und implementiert alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert ohne Fehler und alle Funktionen können ohne Ausführungsfehler ausgeführt werden.		
1	Die Ausgaben des Python-Codes und des COBOL-Codes sind ähnlich, aber nicht identisch.	Der Python-Code übersetzt und implementiert die meisten, aber nicht alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert mit einigen Warnungen, oder es treten bei der Ausführung einige kleinere Fehler auf, die jedoch die grundsätzliche Funktionsfähigkeit nicht beeinträchtigen. Fehler lassen sich mit wenigen zusätzlichen Prompts beheben.		
0	Die Ausgaben des Python-Codes und des COBOL-Codes unterscheiden sich erheblich.	Der Python-Code lässt wesentliche Funktionen, Verfahren oder Operationen des ursprünglichen COBOL-Codes aus oder implementiert sie falsch.	Der Python-Code kompiliert nicht, oder es treten bei der Ausführung schwere Fehler auf, die die Funktionsfähigkeit beeinträchtigen. Fehler lassen sich kaum beheben		
				Total Punkte	6
				Erreichte Punkte	4
				Erreichte Punkte in Prozent	66.66667
	Optimalität				
	Grösse des Programms	**Redundanz**	**Einhaltung von Best Practices**		
2	Der Python-Code ist effizient geschrieben und so gut wie möglich komprimiert. Es gibt eine angemessene Anzahl von Leerzeilen zur Strukturierung.	Der Python-Code vermeidet redundante Operationen oder Wiederholungen. Code-Wiederverwendung wird maximiert.	Der Python-Code hält sich an allgemein anerkannte Best Practices für Python-Programmierung, einschliesslich Einrückung, Benennungskonventionen, Verwendung von Funktionen und Klassen usw.		
1	Der Python-Code ist im Allgemeinen effizient geschrieben, es gibt jedoch Möglichkeiten zur Optimierung.	Der Python-Code enthält einige redundante Operationen oder Wiederholungen, die reduziert werden könnten.	Der Python-Code hält sich grösstenteils an Best Practices, es gibt jedoch einige Verstösse.		
0	Der Python-Code ist ineffizient und zu lang. Es gibt zu viele Leerzeilen.	Der Python-Code ist voller redundanter Operationen oder Wiederholungen.	Der Python-Code verstösst häufig gegen Best Practices.		
				Total Punkte	6
				Erreichte Punkte	5
				Erreichte Punkte in Prozent	83.33333
	Verständlichkeit				
	Lesbarkeit	**Variablen und Funktionen**	**Kommentare**		
2	Der Python-Code ist einfach zu lesen und zu	Variablen und Funktionen haben klare,	Der Code ist gut kommentiert, und die		
1	Der Python-Code ist verständlich, enthält aber unklare Abschnitte.	Die meisten Variablen und Funktionen haben klare, beschreibende Namen, aber einige könnten verbessert werden.	Der Code hat einige nützliche Kommentare, aber es gibt Bereiche, die mehr Kommentierung erfordern oder wo die vorhandenen Kommentare nicht klar sind.		
0	Der Python-Code ist schwer zu lesen und zu verstehen.	Variablen und Funktionen haben unklare oder nicht beschreibende Namen.	Der Code hat wenige oder keine hilfreichen Kommentare.		
				Total Punkte	6
				Erreichte Punkte	5
				Erreichte Punkte in Prozent	83.33333
Kommentar:					
- Ausgabe ist anders, da Varying Loop bis und mit 21 zählt					
- Variablen 1:1 übernommen, jedoch nicht sehr verständlich					
- Verwendet keine Main Methode					

Helloworld2

Kriterien zur Bewertung von ChatGPT (ohne Anweisung)					
Punkte	**Funktionalität**				
	Gleichheit der Ausgaben	**Vollständigkeit der Übersetzung**	**Richtigkeit**		
2	Die Ausgaben des Python-Codes und des COBOL-Codes sind identisch.	Der Python-Code übersetzt und implementiert alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert ohne Fehler und alle Funktionen können ohne Ausführungsfehler ausgeführt werden.		
1	Die Ausgaben des Python-Codes und des COBOL-Codes sind ähnlich, aber nicht identisch.	Der Python-Code übersetzt und implementiert die meisten, aber nicht alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert mit einigen Warnungen, oder es treten bei der Ausführung einige kleinere Fehler auf, die jedoch die grundsätzliche Funktionsfähigkeit nicht beeinträchtigen. Fehler lassen sich mit wenigen zusätzlichen Prompts beheben.		
0	Die Ausgaben des Python-Codes und des COBOL-Codes unterscheiden sich erheblich.	Der Python-Code lässt wesentliche Funktionen, Verfahren oder Operationen des ursprünglichen COBOL-Codes aus oder implementiert sie falsch.	Der Python-Code kompiliert nicht, oder es treten bei der Ausführung schwere Fehler auf, die die Funktionsfähigkeit beeinträchtigen. Fehler lassen sich kaum beheben		
				Total Punkte	6
				Erreichte Punkte	6
				Erreichte Punkte in Prozent	100
	Optimalität				
	Grösse des Programms	**Redundanz**	**Einhaltung von Best Practices**		
2	Der Python-Code ist effizient geschrieben und so gut wie möglich komprimiert. Es gibt eine angemessene Anzahl von Leerzeilen zur Strukturierung.	Der Python-Code vermeidet redundante Operationen oder Wiederholungen. Code-Wiederverwendung wird maximiert.	Der Python-Code hält sich an allgemein anerkannte Best Practices für Python-Programmierung, einschliesslich Einrückung, Benennungskonventionen, Verwendung von Funktionen und Klassen usw.		
1	Der Python-Code ist im Allgemeinen effizient geschrieben, es gibt jedoch Möglichkeiten zur Optimierung.	Der Python-Code enthält einige redundante Operationen oder Wiederholungen, die reduziert werden könnten.	Der Python-Code hält sich grösstenteils an Best Practices, es gibt jedoch einige Verstösse.		
0	Der Python-Code ist ineffizient und zu lang. Es gibt zu viele Leerzeilen.	Der Python-Code ist voller redundanter Operationen oder Wiederholungen.	Der Python-Code verstösst häufig gegen Best Practices.		
				Total Punkte	6
				Erreichte Punkte	6
				Erreichte Punkte in Prozent	100
	Verständlichkeit				
	Lesbarkeit	**Variablen und Funktionen**	**Kommentare**		
2	Der Python-Code ist einfach zu lesen und zu verstehen.	Variablen und Funktionen haben klare, beschreibende Namen, die ihrem Zweck im Code entsprechen.	Der Code ist gut kommentiert, und die Kommentare tragen effektiv zum Verständnis des Codes bei.		
1	Der Python-Code ist verständlich, enthält aber unklare Abschnitte.	Die meisten Variablen und Funktionen haben klare, beschreibende Namen, aber einige könnten verbessert werden.	Der Code hat einige nützliche Kommentare, aber es gibt Bereiche, die mehr Kommentierung erfordern oder wo die vorhandenen Kommentare nicht klar sind.		
0	Der Python-Code ist schwer zu lesen und zu verstehen.	Variablen und Funktionen haben unklare oder nicht beschreibende Namen.	Der Code hat wenige oder keine hilfreichen Kommentare.		
				Total Punkte	6
				Erreichte Punkte	6
				Erreichte Punkte in Prozent	100
Kommentar:					

Kriterien zur Bewertung von ChatGPT (mit Anweisung)					
Punkte	**Funktionalität**				
	Gleichheit der Ausgaben	**Vollständigkeit der Übersetzung**	**Richtigkeit**		
2	Die Ausgaben des Python-Codes und des COBOL-Codes sind identisch.	Der Python-Code übersetzt und implementiert alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert ohne Fehler und alle Funktionen können ohne Ausführungsfehler ausgeführt werden.		
1	Die Ausgaben des Python-Codes und des COBOL-Codes sind ähnlich, aber nicht identisch.	Der Python-Code übersetzt und implementiert die meisten, aber nicht alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert mit einigen Warnungen, oder es treten bei der Ausführung einige kleinere Fehler auf, die jedoch die grundsätzliche Funktionsfähigkeit nicht beeinträchtigen. Fehler lassen sich mit wenigen zusätzlichen Prompts beheben.		
0	Die Ausgaben des Python-Codes und des COBOL-Codes unterscheiden sich erheblich.	Der Python-Code lässt wesentliche Funktionen, Verfahren oder Operationen des ursprünglichen COBOL-Codes aus oder implementiert sie falsch.	Der Python-Code kompiliert nicht, oder es treten bei der Ausführung schwere Fehler auf, die die Funktionsfähigkeit beeinträchtigen. Fehler lassen sich kaum beheben		
				Total Punkte	6
				Erreichte Punkte	6
				Erreichte Punkte in Prozent	100
	Optimalität				
	Grösse des Programms	**Redundanz**	**Einhaltung von Best Practices**		
2	Der Python-Code ist effizient geschrieben und so gut wie möglich komprimiert. Es gibt eine angemessene Anzahl von Leerzeilen zur Strukturierung.	Der Python-Code vermeidet redundante Operationen oder Wiederholungen. Code-Wiederverwendung wird maximiert.	Der Python-Code hält sich an allgemein anerkannte Best Practices für Python-Programmierung, einschliesslich Einrückung, Benennungskonventionen, Verwendung von Funktionen und Klassen usw.		
1	Der Python-Code ist im Allgemeinen effizient geschrieben, es gibt jedoch Möglichkeiten zur Optimierung.	Der Python-Code enthält einige redundante Operationen oder Wiederholungen, die reduziert werden könnten.	Der Python-Code hält sich grösstenteils an Best Practices, es gibt jedoch einige Verstösse.		
0	Der Python-Code ist ineffizient und zu lang. Es gibt zu viele Leerzeilen.	Der Python-Code ist voller redundanter Operationen oder Wiederholungen.	Der Python-Code verstösst häufig gegen Best Practices.		
				Total Punkte	6
				Erreichte Punkte	6
				Erreichte Punkte in Prozent	100
	Verständlichkeit				
	Lesbarkeit	**Variablen und Funktionen**	**Kommentare**		
2	Der Python-Code ist einfach zu lesen und zu verstehen.	Variablen und Funktionen haben klare, beschreibende Namen, die ihrem Zweck im Code entsprechen.	Der Code ist gut kommentiert, und die Kommentare tragen effektiv zum Verständnis des Codes bei.		
1	Der Python-Code ist verständlich, enthält aber unklare Abschnitte.	Die meisten Variablen und Funktionen haben klare, beschreibende Namen, aber einige könnten verbessert werden.	Der Code hat einige nützliche Kommentare, aber es gibt Bereiche, die mehr Kommentierung erfordern oder wo die vorhandenen Kommentare nicht klar sind.		
0	Der Python-Code ist schwer zu lesen und zu verstehen.	Variablen und Funktionen haben unklare oder nicht beschreibende Namen.	Der Code hat wenige oder keine hilfreichen Kommentare.		
				Total Punkte	6
				Erreichte Punkte	6
				Erreichte Punkte in Prozent	100
Kommentar:					

Jsongenerate

Kriterien zur Bewertung von ChatGPT (ohne Anweisung)					
Punkte	**Funktionalität**				
	Gleichheit der Ausgaben	**Vollständigkeit der Übersetzung**	**Richtigkeit**		
2	Die Ausgaben des Python-Codes und des COBOL-Codes sind identisch.	Der Python-Code übersetzt und implementiert alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert ohne Fehler und alle Funktionen können ohne Ausführungsfehler ausgeführt werden.		
1	Die Ausgaben des Python-Codes und des COBOL-Codes sind ähnlich, aber nicht identisch.	Der Python-Code übersetzt und implementiert die meisten, aber nicht alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert mit einigen Warnungen, oder es treten bei der Ausführung einige kleinere Fehler auf, die jedoch die grundsätzliche Funktionsfähigkeit nicht beeinträchtigen. Fehler lassen sich mit wenigen zusätzlichen Prompts beheben.		
0	Die Ausgaben des Python-Codes und des COBOL-Codes unterscheiden sich erheblich.	Der Python-Code lässt wesentliche Funktionen, Verfahren oder Operationen des ursprünglichen COBOL-Codes aus oder implementiert sie falsch.	Der Python-Code kompiliert nicht, oder es treten bei der Ausführung schwere Fehler auf, die die Funktionsfähigkeit beeinträchtigen. Fehler lassen sich kaum beheben		
				Total Punkte	6
				Erreichte Punkte	6
				Erreichte Punkte in Prozent	100
	Optimalität				
	Grösse des Programms	**Redundanz**	**Einhaltung von Best Practices**		
2	Der Python-Code ist effizient geschrieben und so gut wie möglich komprimiert. Es gibt eine angemessene Anzahl von Leerzeilen zur Strukturierung.	Der Python-Code vermeidet redundante Operationen oder Wiederholungen. Code-Wiederverwendung wird maximiert.	Der Python-Code hält sich an allgemein anerkannte Best Practices für Python-Programmierung, einschliesslich Einrückung, Benennungskonventionen, Verwendung von Funktionen und Klassen usw.		
1	Der Python-Code ist im Allgemeinen effizient geschrieben, es gibt jedoch Möglichkeiten zur Optimierung.	Der Python-Code enthält einige redundante Operationen oder Wiederholungen, die reduziert werden könnten.	Der Python-Code hält sich grösstenteils an Best Practices, es gibt jedoch einige Verstösse.		
0	Der Python-Code ist ineffizient und zu lang. Es gibt zu viele Leerzeilen.	Der Python-Code ist voller redundanter Operationen oder Wiederholungen.	Der Python-Code verstösst häufig gegen Best Practices.		
				Total	6
				Erreichte Punkte	5
				Erreichte Punkte in Prozent	83.33333
	Verständlichkeit				
	Lesbarkeit	**Variablen und Funktionen**	**Kommentare**		
2	Der Python-Code ist einfach zu lesen und zu verstehen.	Variablen und Funktionen haben klare, beschreibende Namen, die ihrem Zweck im Code entsprechen.	Der Code ist gut kommentiert, und die Kommentare tragen effektiv zum Verständnis des Codes bei.		
1	Der Python-Code ist verständlich, enthält aber unklare Abschnitte.	Die meisten Variablen und Funktionen haben klare, beschreibende Namen, aber einige könnten verbessert werden.	Der Code hat einige nützliche Kommentare, aber es gibt Bereiche, die mehr Kommentierung erfordern oder wo die vorhandenen Kommentare nicht klar sind.		
0	Der Python-Code ist schwer zu lesen und zu verstehen.	Variablen und Funktionen haben unklare oder nicht beschreibende Namen.	Der Code hat wenige oder keine hilfreichen Kommentare.		
				Total Punkte	6
				Erreichte Punkte	4
				Erreichte Punkte in Prozent	66.66667
Kommentar:					
- Ausgabe in Python verbessert. Zeigt Key-Value-Paare in Json Format an und gibt character count aus, der im COBOL Programm fehlt. Ansonsten identisch					
- Try Except könnte spezifischer sein. Diese Exception fängt diverse Fehler ab, z.B. dass Datei nicht vorhanden ist, Format nicht stimmt etc. Könnte daran liegen, dass es in COBOL keine spezifischere Exception gibt					
- Nur ein Kommentar enthalten					
- Benennung von Funktion und Klasse könnte eindeutiger sein					
- Verwendet Main Methode					

Kriterien zur Bewertung von ChatGPT (mit Anweisung)					
Punkte	**Funktionalität**				
	Gleichheit der Ausgaben	**Vollständigkeit der Übersetzung**	**Richtigkeit**		
2	Die Ausgaben des Python-Codes und des COBOL-Codes sind identisch.	Der Python-Code übersetzt und implementiert alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert ohne Fehler und alle Funktionen können ohne Ausführungsfehler ausgeführt werden.		
1	Die Ausgaben des Python-Codes und des COBOL-Codes sind ähnlich, aber nicht identisch.	Der Python-Code übersetzt und implementiert die meisten, aber nicht alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert mit einigen Warnungen, oder es treten bei der Ausführung einige kleinere Fehler auf, die jedoch die grundsätzliche Funktionsfähigkeit nicht beeinträchtigen. Fehler lassen sich mit wenigen zusätzlichen Prompts beheben.		
0	Die Ausgaben des Python-Codes und des COBOL-Codes unterscheiden sich erheblich.	Der Python-Code lässt wesentliche Funktionen, Verfahren oder Operationen des ursprünglichen COBOL-Codes aus oder implementiert sie falsch.	Der Python-Code kompiliert nicht, oder es treten bei der Ausführung schwere Fehler auf, die die Funktionsfähigkeit beeinträchtigen. Fehler lassen sich kaum beheben		
				Total Punkte	6
				Erreichte Punkte	6
				Erreichte Punkte in Prozent	100
	Optimalität				
	Grösse des Programms	**Redundanz**	**Einhaltung von Best Practices**		
2	Der Python-Code ist effizient geschrieben und so gut wie möglich komprimiert. Es gibt eine angemessene Anzahl von Leerzeilen zur Strukturierung.	Der Python-Code vermeidet redundante Operationen oder Wiederholungen. Code-Wiederverwendung wird maximiert.	Der Python-Code hält sich an allgemein anerkannte Best Practices für Python-Programmierung, einschliesslich Einrückung, Benennungskonventionen, Verwendung von Funktionen und Klassen usw.		
1	Der Python-Code ist im Allgemeinen effizient geschrieben, es gibt jedoch Möglichkeiten zur Optimierung.	Der Python-Code enthält einige redundante Operationen oder Wiederholungen, die reduziert werden könnten.	Der Python-Code hält sich grösstenteils an Best Practices, es gibt jedoch einige Verstösse.		
0	Der Python-Code ist ineffizient und zu lang. Es gibt zu viele Leerzeilen.	Der Python-Code ist voller redundanter Operationen oder Wiederholungen.	Der Python-Code verstösst häufig gegen Best Practices.		
				Total	6
				Erreichte Punkte	4
				Erreichte Punkte in Prozent	66.66667
	Verständlichkeit				
	Lesbarkeit	**Variablen und Funktionen**	**Kommentare**		
2	Der Python-Code ist einfach zu lesen und zu verstehen.	Variablen und Funktionen haben klare, beschreibende Namen, die ihrem Zweck im Code entsprechen.	Der Code ist gut kommentiert, und die Kommentare tragen effektiv zum Verständnis des Codes bei.		
1	Der Python-Code ist verständlich, enthält aber unklare Abschnitte.	Die meisten Variablen und Funktionen haben klare, beschreibende Namen, aber einige könnten verbessert werden.	Der Code hat einige nützliche Kommentare, aber es gibt Bereiche, die mehr Kommentierung erfordern oder wo die vorhandenen Kommentare nicht klar sind.		
0	Der Python-Code ist schwer zu lesen und zu verstehen.	Variablen und Funktionen haben unklare oder nicht beschreibende Namen.	Der Code hat wenige oder keine hilfreichen Kommentare.		
				Total Punkte	6
				Erreichte Punkte	5
				Erreichte Punkte in Prozent	83.33333
Kommentar:					
- Werte für Variablen 6-8 hätten bereits bei der Definition der Variablen erfolgen können, anstatt in Zeile 12-14					
- Ansonsten ähnlich zu bewerten wie unsupervised					

Notenrechner

Kriterien zur Bewertung von ChatGPT (ohne Anweisung)					
Punkte	**Funktionalität**				
	Gleichheit der Ausgaben	**Vollständigkeit der Übersetzung**	**Richtigkeit**		
2	Die Ausgaben des Python-Codes und des COBOL-Codes sind identisch.	Der Python-Code übersetzt und implementiert alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert ohne Fehler und alle Funktionen können ohne Ausführungsfehler ausgeführt werden.		
1	Die Ausgaben des Python-Codes und des COBOL-Codes sind ähnlich, aber nicht identisch.	Der Python-Code übersetzt und implementiert die meisten, aber nicht alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert mit einigen Warnungen, oder es treten bei der Ausführung einige kleinere Fehler auf, die jedoch die grundsätzliche Funktionsfähigkeit nicht beeinträchtigen. Fehler lassen sich mit wenigen zusätzlichen Prompts beheben.		
0	Die Ausgaben des Python-Codes und des COBOL-Codes unterscheiden sich erheblich.	Der Python-Code lässt wesentliche Funktionen, Verfahren oder Operationen des ursprünglichen COBOL-Codes aus oder implementiert sie falsch.	Der Python-Code kompiliert nicht, oder es treten bei der Ausführung schwere Fehler auf, die die Funktionsfähigkeit beeinträchtigen. Fehler lassen sich kaum beheben		
				Total Punkte	6
				Erreichte Punkte	2
				Erreichte Punkte in Prozent	33.33333
	Optimalität				
	Grösse des Programms	**Redundanz**	**Einhaltung von Best Practices**		
2	Der Python-Code ist effizient geschrieben und so gut wie möglich komprimiert. Es gibt eine angemessene Anzahl von Leerzeilen zur Strukturierung.	Der Python-Code vermeidet redundante Operationen oder Wiederholungen. Code-Wiederverwendung wird maximiert.	Der Python-Code hält sich an allgemein anerkannte Best Practices für Python-Programmierung, einschliesslich Einrückung, Benennungskonventionen, Verwendung von Funktionen und Klassen usw.		
1	Der Python-Code ist im Allgemeinen effizient geschrieben, es gibt jedoch Möglichkeiten zur Optimierung.	Der Python-Code enthält einige redundante Operationen oder Wiederholungen, die reduziert werden könnten.	Der Python-Code hält sich grösstenteils an Best Practices, es gibt jedoch einige Verstösse.		
0	Der Python-Code ist ineffizient und zu lang. Es gibt zu viele Leerzeilen.	Der Python-Code ist voller redundanter Operationen oder Wiederholungen.	Der Python-Code verstösst häufig gegen Best Practices.		
				Total	6
				Erreichte Punkte	0
				Erreichte Punkte in Prozent	0
	Verständlichkeit				
	Lesbarkeit	**Variablen und Funktionen**	**Kommentare**		
2	Der Python-Code ist einfach zu lesen und zu verstehen.	Variablen und Funktionen haben klare, beschreibende Namen, die ihrem Zweck im Code entsprechen.	Der Code ist gut kommentiert, und die Kommentare tragen effektiv zum Verständnis des Codes bei.		
1	Der Python-Code ist verständlich, enthält aber unklare Abschnitte.	Die meisten Variablen und Funktionen haben klare, beschreibende Namen, aber einige könnten verbessert werden.	Der Code hat einige nützliche Kommentare, aber es gibt Bereiche, die mehr Kommentierung erfordern oder wo die vorhandenen Kommentare nicht klar sind.		
0	Der Python-Code ist schwer zu lesen und zu verstehen.	Variablen und Funktionen haben unklare oder nicht beschreibende Namen.	Der Code hat wenige oder keine hilfreichen Kommentare.		
				Total Punkte	6
				Erreichte Punkte	0
				Erreichte Punkte in Prozent	0
Kommentar:					
- Code produziert komplett andere Ausgaben					
- Folgt keinen Best Practices z.B. durch Verwendung von globalen Variablen wie in Zeile 17					
- Innefizziente If-Abfragen, die teilweise nicht alle Cases abdecken (bsp. Zeile 66)					
- Keine Kommentare, aufgeblasener und unübersichtlicher Code					

Kriterien zur Bewertung von ChatGPT (mit Anweisung)					
Punkte	**Funktionalität**				
	Gleichheit der Ausgaben	**Vollständigkeit der Übersetzung**	**Richtigkeit**		
2	Die Ausgaben des Python-Codes und des COBOL-Codes sind identisch.	Der Python-Code übersetzt und implementiert alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert ohne Fehler und alle Funktionen können ohne Ausführungsfehler ausgeführt werden.		
1	Die Ausgaben des Python-Codes und des COBOL-Codes sind ähnlich, aber nicht identisch.	Der Python-Code übersetzt und implementiert die meisten, aber nicht alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert mit einigen Warnungen, oder es treten bei der Ausführung einige kleinere Fehler auf, die jedoch die grundsätzliche Funktionsfähigkeit nicht beeinträchtigen. Fehler lassen sich mit wenigen zusätzlichen Prompts beheben.		
0	Die Ausgaben des Python-Codes und des COBOL-Codes unterscheiden sich erheblich.	Der Python-Code lässt wesentliche Funktionen, Verfahren oder Operationen des ursprünglichen COBOL-Codes aus oder implementiert sie falsch.	Der Python-Code kompiliert nicht, oder es treten bei der Ausführung schwere Fehler auf, die die Funktionsfähigkeit beeinträchtigen. Fehler lassen sich kaum beheben		
				Total Punkte	6
				Erreichte Punkte	5
				Erreichte Punkte in Prozent	83.33333
Optimalität					
	Grösse des Programms	**Redundanz**	**Einhaltung von Best Practices**		
2	Der Python-Code ist effizient geschrieben und so gut wie möglich komprimiert. Es gibt eine angemessene Anzahl von Leerzeilen zur Strukturierung.	Der Python-Code vermeidet redundante Operationen oder Wiederholungen. Code-Wiederverwendung wird maximiert.	Der Python-Code hält sich an allgemein anerkannte Best Practices für Python-Programmierung, einschliesslich Einrückung, Benennungskonventionen, Verwendung von Funktionen und Klassen usw.		
1	Der Python-Code ist im Allgemeinen effizient geschrieben, es gibt jedoch Möglichkeiten zur Optimierung.	Der Python-Code enthält einige redundante Operationen oder Wiederholungen, die reduziert werden könnten.	Der Python-Code hält sich grösstenteils an Best Practices, es gibt jedoch einige Verstösse.		
0	Der Python-Code ist ineffizient und zu lang. Es gibt zu viele Leerzeilen.	Der Python-Code ist voller redundanter Operationen oder Wiederholungen.	Der Python-Code verstösst häufig gegen Best Practices.		
				Total	6
				Erreichte Punkte	0
				Erreichte Punkte in Prozent	0
Verständlichkeit					
	Lesbarkeit	**Variablen und Funktionen**	**Kommentare**		
2	Der Python-Code ist einfach zu lesen und zu verstehen.	Variablen und Funktionen haben klare, beschreibende Namen, die ihrem Zweck im Code entsprechen.	Der Code ist gut kommentiert, und die Kommentare tragen effektiv zum Verständnis des Codes bei.		
1	Der Python-Code ist verständlich, enthält aber unklare Abschnitte.	Die meisten Variablen und Funktionen haben klare, beschreibende Namen, aber einige könnten verbessert werden.	Der Code hat einige nützliche Kommentare, aber es gibt Bereiche, die mehr Kommentierung erfordern oder wo die vorhandenen Kommentare nicht klar sind.		
0	Der Python-Code ist schwer zu lesen und zu verstehen.	Variablen und Funktionen haben unklare oder nicht beschreibende Namen.	Der Code hat wenige oder keine hilfreichen Kommentare.		
				Total Punkte	6
				Erreichte Punkte	0
				Erreichte Punkte in Prozent	0
Kommentar:					
- Code weist einen Fehler in Zeile 70 auf, kann jedoch mit einem Prompt korrigiert und dann kompiliert werden					
- Code nach wie vor sehr ineffizient und unleserlich					
- Best Practices werden nicht eingehalten (globale Variablen, ...) und macht den Code dadurch sehr unschön strukturiert und kompliziert.					
- Enthält trotz Anweisung keine Kommentare, die zur Verständlichkeit beitragen.					

Npkcalculator

Kriterien zur Bewertung von ChatGPT (ohne Anweisung)					
Punkte	**Funktionalität**				
	Gleichheit der Ausgaben	**Vollständigkeit der Übersetzung**	**Richtigkeit**		
2	Die Ausgaben des Python-Codes und des COBOL-Codes sind identisch.	Der Python-Code übersetzt und implementiert alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert ohne Fehler und alle Funktionen können ohne Ausführungsfehler ausgeführt werden.		
1	Die Ausgaben des Python-Codes und des COBOL-Codes sind ähnlich, aber nicht identisch.	Der Python-Code übersetzt und implementiert die meisten, aber nicht alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert mit einigen Warnungen, oder es treten bei der Ausführung einige kleinere Fehler auf, die jedoch die grundsätzliche Funktionsfähigkeit nicht beeinträchtigen. Fehler lassen sich mit wenigen zusätzlichen Prompts beheben.		
0	Die Ausgaben des Python-Codes und des COBOL-Codes unterscheiden sich erheblich.	Der Python-Code lässt wesentliche Funktionen, Verfahren oder Operationen des ursprünglichen COBOL-Codes aus oder implementiert sie falsch.	Der Python-Code kompiliert nicht, oder es treten bei der Ausführung schwere Fehler auf, die die Funktionsfähigkeit beeinträchtigen. Fehler lassen sich kaum beheben		
				Total Punkte	6
				Erreichte Punkte	6
				Erreichte Punkte in Prozent	100
	Optimalität				
	Grösse des Programms	**Redundanz**	**Einhaltung von Best Practices**		
2	Der Python-Code ist effizient geschrieben und so gut wie möglich komprimiert. Es gibt eine angemessene Anzahl von Leerzeilen zur Strukturierung.	Der Python-Code vermeidet redundante Operationen oder Wiederholungen. Code-Wiederverwendung wird maximiert.	Der Python-Code hält sich an allgemein anerkannte Best Practices für Python-Programmierung, einschliesslich Einrückung, Benennungskonventionen, Verwendung von Funktionen und Klassen usw.		
1	Der Python-Code ist im Allgemeinen effizient geschrieben, es gibt jedoch Möglichkeiten zur Optimierung.	Der Python-Code enthält einige redundante Operationen oder Wiederholungen, die reduziert werden könnten.	Der Python-Code hält sich grösstenteils an Best Practices, es gibt jedoch einige Verstösse.		
0	Der Python-Code ist ineffizient und zu lang. Es gibt zu viele Leerzeilen.	Der Python-Code ist voller redundanter Operationen oder Wiederholungen.	Der Python-Code verstösst häufig gegen Best Practices.		
				Total	6
				Erreichte Punkte	5
				Erreichte Punkte in Prozent	83.33333
	Verständlichkeit				
	Lesbarkeit	**Variablen und Funktionen**	**Kommentare**		
2	Der Python-Code ist einfach zu lesen und zu verstehen.	Variablen und Funktionen haben klare, beschreibende Namen, die ihrem Zweck im Code entsprechen.	Der Code ist gut kommentiert, und die Kommentare tragen effektiv zum Verständnis des Codes bei.		
1	Der Python-Code ist verständlich, enthält aber unklare Abschnitte.	Die meisten Variablen und Funktionen haben klare, beschreibende Namen, aber einige könnten verbessert werden.	Der Code hat einige nützliche Kommentare, aber es gibt Bereiche, die mehr Kommentierung erfordern oder wo die vorhandenen Kommentare nicht klar sind.		
0	Der Python-Code ist schwer zu lesen und zu verstehen.	Variablen und Funktionen haben unklare oder nicht beschreibende Namen.	Der Code hat wenige oder keine hilfreichen Kommentare.		
				Total Punkte	6
				Erreichte Punkte	4
				Erreichte Punkte in Prozent	66.66667
Kommentar:					
- Bildet COBOL Programm beinahe 1:1 ab. Legt mehr Wert auf Nachbau des Eingabeprogramms, als darauf das optimale Programm zu bauen.					
- Enthält keine Kommentare					
- Zweckentfremdet Variable op in Zeile 52, indem er dein eingegebenen Operator überschreibt (wiederholt sich im Loop)					

Kriterien zur Bewertung von ChatGPT (mit Anweisung)					
Punkte	**Funktionalität**				
	Gleichheit der Ausgaben	**Vollständigkeit der Übersetzung**	**Richtigkeit**		
2	Die Ausgaben des Python-Codes und des COBOL-Codes sind identisch.	Der Python-Code übersetzt und implementiert alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert ohne Fehler und alle Funktionen können ohne Ausführungsfehler ausgeführt werden.		
1	Die Ausgaben des Python-Codes und des COBOL-Codes sind ähnlich, aber nicht identisch.	Der Python-Code übersetzt und implementiert die meisten, aber nicht alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert mit einigen Warnungen, oder es treten bei der Ausführung einige kleinere Fehler auf, die jedoch die grundsätzliche Funktionsfähigkeit nicht beeinträchtigen. Fehler lassen sich mit wenigen zusätzlichen Prompts beheben.		
0	Die Ausgaben des Python-Codes und des COBOL-Codes unterscheiden sich erheblich.	Der Python-Code lässt wesentliche Funktionen, Verfahren oder Operationen des ursprünglichen COBOL-Codes aus oder implementiert sie falsch.	Der Python-Code kompiliert nicht, oder es treten bei der Ausführung schwere Fehler auf, die die Funktionsfähigkeit beeinträchtigen. Fehler lassen sich kaum beheben		
				Total Punkte	6
				Erreichte Punkte	6
				Erreichte Punkte in Prozent	100
	Optimalität				
	Grösse des Programms	**Redundanz**	**Einhaltung von Best Practices**		
2	Der Python-Code ist effizient geschrieben und so gut wie möglich komprimiert. Es gibt eine angemessene Anzahl von Leerzeilen zur Strukturierung.	Der Python-Code vermeidet redundante Operationen oder Wiederholungen. Code-Wiederverwendung wird maximiert.	Der Python-Code hält sich an allgemein anerkannte Best Practices für Python-Programmierung, einschliesslich Einrückung, Benennungskonventionen, Verwendung von Funktionen und Klassen usw.		
1	Der Python-Code ist im Allgemeinen effizient geschrieben, es gibt jedoch Möglichkeiten zur Optimierung.	Der Python-Code enthält einige redundante Operationen oder Wiederholungen, die reduziert werden könnten.	Der Python-Code hält sich grösstenteils an Best Practices, es gibt jedoch einige Verstösse.		
0	Der Python-Code ist ineffizient und zu lang. Es gibt zu viele Leerzeilen.	Der Python-Code ist voller redundanter Operationen oder Wiederholungen.	Der Python-Code verstösst häufig gegen Best Practices.		
				Total	6
				Erreichte Punkte	3
				Erreichte Punkte in Prozent	50
	Verständlichkeit				
	Lesbarkeit	**Variablen und Funktionen**	**Kommentare**		
2	Der Python-Code ist einfach zu lesen und zu verstehen.	Variablen und Funktionen haben klare, beschreibende Namen, die ihrem Zweck im Code entsprechen.	Der Code ist gut kommentiert, und die Kommentare tragen effektiv zum Verständnis des Codes bei.		
1	Der Python-Code ist verständlich, enthält aber unklare Abschnitte.	Die meisten Variablen und Funktionen haben klare, beschreibende Namen, aber einige könnten verbessert werden.	Der Code hat einige nützliche Kommentare, aber es gibt Bereiche, die mehr Kommentierung erfordern oder wo die vorhandenen Kommentare nicht klar sind.		
0	Der Python-Code ist schwer zu lesen und zu verstehen.	Variablen und Funktionen haben unklare oder nicht beschreibende Namen.	Der Code hat wenige oder keine hilfreichen Kommentare.		
				Total Punkte	6
				Erreichte Punkte	4
				Erreichte Punkte in Prozent	66.66667
Kommentar:					
- Verwendet globale Variablen, anstatt return zu verwenden in Funktionen					
- Kommentare werden inkonsistent verwendet und sind wenig aussagekräftig					
- Code ist sehr lange und ineffizient geschrieben, büsst darum an Verständlichkeit ein					

Numval

Kriterien zur Bewertung von ChatGPT (ohne Anweisung)					
Punkte	**Funktionalität**				
	Gleichheit der Ausgaben	**Vollständigkeit der Übersetzung**	**Richtigkeit**		
2	Die Ausgaben des Python-Codes und des COBOL-Codes sind identisch.	Der Python-Code übersetzt und implementiert alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert ohne Fehler und alle Funktionen können ohne Ausführungsfehler ausgeführt werden.		
1	Die Ausgaben des Python-Codes und des COBOL-Codes sind ähnlich, aber nicht identisch.	Der Python-Code übersetzt und implementiert die meisten, aber nicht alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert mit einigen Warnungen, oder es treten bei der Ausführung einige kleinere Fehler auf, die jedoch die grundsätzliche Funktionsfähigkeit nicht beeinträchtigen. Fehler lassen sich mit wenigen zusätzlichen Prompts beheben.		
0	Die Ausgaben des Python-Codes und des COBOL-Codes unterscheiden sich erheblich.	Der Python-Code lässt wesentliche Funktionen, Verfahren oder Operationen des ursprünglichen COBOL-Codes aus oder implementiert sie falsch.	Der Python-Code kompiliert nicht, oder es treten bei der Ausführung schwere Fehler auf, die die Funktionsfähigkeit beeinträchtigen. Fehler lassen sich kaum beheben		
				Total Punkte	6
				Erreichte Punkte	5
				Erreichte Punkte in Prozent	83.33333
	Optimalität				
	Grösse des Programms	**Redundanz**	**Einhaltung von Best Practices**		
2	Der Python-Code ist effizient geschrieben und so gut wie möglich komprimiert. Es gibt eine angemessene Anzahl von Leerzeilen zur Strukturierung.	Der Python-Code vermeidet redundante Operationen oder Wiederholungen. Code-Wiederverwendung wird maximiert.	Der Python-Code hält sich an allgemein anerkannte Best Practices für Python-Programmierung, einschliesslich Einrückung, Benennungskonventionen, Verwendung von Funktionen und Klassen usw.		
1	Der Python-Code ist im Allgemeinen effizient geschrieben, es gibt jedoch Möglichkeiten zur Optimierung.	Der Python-Code enthält einige redundante Operationen oder Wiederholungen, die reduziert werden könnten.	Der Python-Code hält sich grösstenteils an Best Practices, es gibt jedoch einige Verstösse.		
0	Der Python-Code ist ineffizient und zu lang. Es gibt zu viele Leerzeilen.	Der Python-Code ist voller redundanter Operationen oder Wiederholungen.	Der Python-Code verstösst häufig gegen Best Practices.		
				Total Punkte	6
				Erreichte Punkte	5
				Erreichte Punkte in Prozent	83.33333
	Verständlichkeit				
	Lesbarkeit	**Variablen und Funktionen**	**Kommentare**		
2	Der Python-Code ist einfach zu lesen und zu	Variablen und Funktionen haben klare,	Der Code ist gut kommentiert, und die		
1	Der Python-Code ist verständlich, enthält aber unklare Abschnitte.	Die meisten Variablen und Funktionen haben klare, beschreibende Namen, aber einige könnten verbessert werden.	Der Code hat einige nützliche Kommentare, aber es gibt Bereiche, die mehr Kommentierung erfordern oder wo die vorhandenen Kommentare nicht klar sind.		
0	Der Python-Code ist schwer zu lesen und zu verstehen.	Variablen und Funktionen haben unklare oder nicht beschreibende Namen.	Der Code hat wenige oder keine hilfreichen Kommentare.		
				Total Punkte	6
				Erreichte Punkte	4
				Erreichte Punkte in Prozent	66.66667

Kommentar:
- Variablen und Funktionsnamen werden 1:1 übernommen, könnte optimiert sein (v.a. Funktionsname main_procedure)
- Kommentar erklärt, warum Float verwendet wurde. Ausgabe ändert im Vergleich zum Ausgansprogramm leicht, da im COBOL Code die Ausgabe rein numerisch ist. Für exakt gleiches Verhalten, hätte hier um die Variable ws_total auch ein int() gesetzt werden müssen.
- Verwendet keine Main Methode

Kriterien zur Bewertung von ChatGPT (mit Anweisung)					
Punkte	**Funktionalität**				
	Gleichheit der Ausgaben	**Vollständigkeit der Übersetzung**	**Richtigkeit**		
2	Die Ausgaben des Python-Codes und des COBOL-Codes sind identisch.	Der Python-Code übersetzt und implementiert alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert ohne Fehler und alle Funktionen können ohne Ausführungsfehler ausgeführt werden.		
1	Die Ausgaben des Python-Codes und des COBOL-Codes sind ähnlich, aber nicht identisch.	Der Python-Code übersetzt und implementiert die meisten, aber nicht alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert mit einigen Warnungen, oder es treten bei der Ausführung einige kleinere Fehler auf, die jedoch die grundsätzliche Funktionsfähigkeit nicht beeinträchtigen. Fehler lassen sich mit wenigen zusätzlichen Prompts beheben.		
0	Die Ausgaben des Python-Codes und des COBOL-Codes unterscheiden sich erheblich.	Der Python-Code lässt wesentliche Funktionen, Verfahren oder Operationen des ursprünglichen COBOL-Codes aus oder implementiert sie falsch.	Der Python-Code kompiliert nicht, oder es treten bei der Ausführung schwere Fehler auf, die die Funktionsfähigkeit beeinträchtigen. Fehler lassen sich kaum beheben		
				Total Punkte	6
				Erreichte Punkte	5
				Erreichte Punkte in Prozent	83.33333
	Optimalität				
	Grösse des Programms	**Redundanz**	**Einhaltung von Best Practices**		
2	Der Python-Code ist effizient geschrieben und so gut wie möglich komprimiert. Es gibt eine angemessene Anzahl von Leerzeilen zur Strukturierung.	Der Python-Code vermeidet redundante Operationen oder Wiederholungen. Code-Wiederverwendung wird maximiert.	Der Python-Code hält sich an allgemein anerkannte Best Practices für Python-Programmierung, einschliesslich Einrückung, Benennungskonventionen, Verwendung von Funktionen und Klassen usw.		
1	Der Python-Code ist im Allgemeinen effizient geschrieben, es gibt jedoch Möglichkeiten zur Optimierung.	Der Python-Code enthält einige redundante Operationen oder Wiederholungen, die reduziert werden könnten.	Der Python-Code hält sich grösstenteils an Best Practices, es gibt jedoch einige Verstösse.		
0	Der Python-Code ist ineffizient und zu lang. Es gibt zu viele Leerzeilen.	Der Python-Code ist voller redundanter Operationen oder Wiederholungen.	Der Python-Code verstösst häufig gegen Best Practices.		
				Total Punkte	6
				Erreichte Punkte	6
				Erreichte Punkte in Prozent	100
	Verständlichkeit				
	Lesbarkeit	**Variablen und Funktionen**	**Kommentare**		
2	Der Python-Code ist einfach zu lesen und zu	Variablen und Funktionen haben klare,	Der Code ist gut kommentiert, und die		
1	Der Python-Code ist verständlich, enthält aber unklare Abschnitte.	Die meisten Variablen und Funktionen haben klare, beschreibende Namen, aber einige könnten verbessert werden.	Der Code hat einige nützliche Kommentare, aber es gibt Bereiche, die mehr Kommentierung erfordern oder wo die vorhandenen Kommentare nicht klar sind.		
0	Der Python-Code ist schwer zu lesen und zu verstehen.	Variablen und Funktionen haben unklare oder nicht beschreibende Namen.	Der Code hat wenige oder keine hilfreichen Kommentare.		
				Total Punkte	6
				Erreichte Punkte	5
				Erreichte Punkte in Prozent	83.33333
Kommentar:					
- Verwendet Main Methode					
- Variablen sind beide Male als Float ausgegeben, verhalten unterscheidet sich dadurch leicht vom ursprünglichen Programm, hätte int() um ws_total setzen müssen.					
- Variablen- und Funktionsnamen 1:1 übernommen, nicht optimiert					

Payroll

Kriterien zur Bewertung von ChatGPT (ohne Anweisung)					
Punkte	**Funktionalität**				
	Gleichheit der Ausgaben	**Vollständigkeit der Übersetzung**	**Richtigkeit**		
2	Die Ausgaben des Python-Codes und des COBOL-Codes sind identisch.	Der Python-Code übersetzt und implementiert alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert ohne Fehler und alle Funktionen können ohne Ausführungsfehler ausgeführt werden.		
1	Die Ausgaben des Python-Codes und des COBOL-Codes sind ähnlich, aber nicht identisch.	Der Python-Code übersetzt und implementiert die meisten, aber nicht alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert mit einigen Warnungen, oder es treten bei der Ausführung einige kleinere Fehler auf, die jedoch die grundsätzliche Funktionsfähigkeit nicht beeinträchtigen. Fehler lassen sich mit wenigen zusätzlichen Prompts beheben.		
0	Die Ausgaben des Python-Codes und des COBOL-Codes unterscheiden sich erheblich.	Der Python-Code lässt wesentliche Funktionen, Verfahren oder Operationen des ursprünglichen COBOL-Codes aus oder implementiert sie falsch.	Der Python-Code kompiliert nicht, oder es treten bei der Ausführung schwere Fehler auf, die die Funktionsfähigkeit beeinträchtigen. Fehler lassen sich kaum beheben		
				Total Punkte	6
				Erreichte Punkte	6
				Erreichte Punkte in Prozent	100
	Optimalität				
	Grösse des Programms	**Redundanz**	**Einhaltung von Best Practices**		
2	Der Python-Code ist effizient geschrieben und so gut wie möglich komprimiert. Es gibt eine angemessene Anzahl von Leerzeilen zur Strukturierung.	Der Python-Code vermeidet redundante Operationen oder Wiederholungen. Code-Wiederverwendung wird maximiert.	Der Python-Code hält sich an allgemein anerkannte Best Practices für Python-Programmierung, einschliesslich Einrückung, Benennungskonventionen, Verwendung von Funktionen und Klassen usw.		
1	Der Python-Code ist im Allgemeinen effizient geschrieben, es gibt jedoch Möglichkeiten zur Optimierung.	Der Python-Code enthält einige redundante Operationen oder Wiederholungen, die reduziert werden könnten.	Der Python-Code hält sich grösstenteils an Best Practices, es gibt jedoch einige Verstösse.		
0	Der Python-Code ist ineffizient und zu lang. Es gibt zu viele Leerzeilen.	Der Python-Code ist voller redundanter Operationen oder Wiederholungen.	Der Python-Code verstösst häufig gegen Best Practices.		
				Total Punkte	6
				Erreichte Punkte	5
				Erreichte Punkte in Prozent	83.33333
	Verständlichkeit				
	Lesbarkeit	**Variablen und Funktionen**	**Kommentare**		
2	Der Python-Code ist einfach zu lesen und zu	Variablen und Funktionen haben klare,	Der Code ist gut kommentiert, und die		
1	Der Python-Code ist verständlich, enthält aber unklare Abschnitte.	Die meisten Variablen und Funktionen haben klare, beschreibende Namen, aber einige könnten verbessert werden.	Der Code hat einige nützliche Kommentare, aber es gibt Bereiche, die mehr Kommentierung erfordern oder wo die vorhandenen Kommentare nicht klar sind.		
0	Der Python-Code ist schwer zu lesen und zu verstehen.	Variablen und Funktionen haben unklare oder nicht beschreibende Namen.	Der Code hat wenige oder keine hilfreichen Kommentare.		
				Total Punkte	6
				Erreichte Punkte	4
				Erreichte Punkte in Prozent	66.66667
Kommentar:					
- Main Methode fehlt					
- Keine Kommentare enthalten					
- Funktionalität identisch					

Kriterien zur Bewertung von ChatGPT (mit Anweisung)					
Punkte	**Funktionalität**				
	Gleichheit der Ausgaben	**Vollständigkeit der Übersetzung**	**Richtigkeit**		
2	Die Ausgaben des Python-Codes und des COBOL-Codes sind identisch.	Der Python-Code übersetzt und implementiert alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert ohne Fehler und alle Funktionen können ohne Ausführungsfehler ausgeführt werden.		
1	Die Ausgaben des Python-Codes und des COBOL-Codes sind ähnlich, aber nicht identisch.	Der Python-Code übersetzt und implementiert die meisten, aber nicht alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert mit einigen Warnungen, oder es treten bei der Ausführung einige kleinere Fehler auf, die jedoch die grundsätzliche Funktionsfähigkeit nicht beeinträchtigen. Fehler lassen sich mit wenigen zusätzlichen Prompts beheben.		
0	Die Ausgaben des Python-Codes und des COBOL-Codes unterscheiden sich erheblich.	Der Python-Code lässt wesentliche Funktionen, Verfahren oder Operationen des ursprünglichen COBOL-Codes aus oder implementiert sie falsch.	Der Python-Code kompiliert nicht, oder es treten bei der Ausführung schwere Fehler auf, die die Funktionsfähigkeit beeinträchtigen. Fehler lassen sich kaum beheben		
				Total Punkte	6
				Erreichte Punkte	6
				Erreichte Punkte in Prozent	100
Optimalität					
	Grösse des Programms	**Redundanz**	**Einhaltung von Best Practices**		
2	Der Python-Code ist effizient geschrieben und so gut wie möglich komprimiert. Es gibt eine angemessene Anzahl von Leerzeilen zur Strukturierung.	Der Python-Code vermeidet redundante Operationen oder Wiederholungen. Code-Wiederverwendung wird maximiert.	Der Python-Code hält sich an allgemein anerkannte Best Practices für Python-Programmierung, einschliesslich Einrückung, Benennungskonventionen, Verwendung von Funktionen und Klassen usw.		
1	Der Python-Code ist im Allgemeinen effizient geschrieben, es gibt jedoch Möglichkeiten zur Optimierung.	Der Python-Code enthält einige redundante Operationen oder Wiederholungen, die reduziert werden könnten.	Der Python-Code hält sich grösstenteils an Best Practices, es gibt jedoch einige Verstösse.		
0	Der Python-Code ist ineffizient und zu lang. Es gibt zu viele Leerzeilen.	Der Python-Code ist voller redundanter Operationen oder Wiederholungen.	Der Python-Code verstösst häufig gegen Best Practices.		
				Total Punkte	6
				Erreichte Punkte	6
				Erreichte Punkte in Prozent	100
Verständlichkeit					
	Lesbarkeit	**Variablen und Funktionen**	**Kommentare**		
2	Der Python-Code ist einfach zu lesen und zu	Variablen und Funktionen haben klare,	Der Code ist gut kommentiert, und die		
1	Der Python-Code ist verständlich, enthält aber unklare Abschnitte.	Die meisten Variablen und Funktionen haben klare, beschreibende Namen, aber einige könnten verbessert werden.	Der Code hat einige nützliche Kommentare, aber es gibt Bereiche, die mehr Kommentierung erfordern oder wo die vorhandenen Kommentare nicht klar sind.		
0	Der Python-Code ist schwer zu lesen und zu verstehen.	Variablen und Funktionen haben unklare oder nicht beschreibende Namen.	Der Code hat wenige oder keine hilfreichen Kommentare.		
				Total Punkte	6
				Erreichte Punkte	6
				Erreichte Punkte in Prozent	100
Kommentar:					
- Identische Ausgabe					
- Beachtet Best Practices (Main Methode, ...)					
- Fügt Kommentare hinzu wo nötig					

Randombingo

Kriterien zur Bewertung von ChatGPT (ohne Anweisung)					
Punkte	**Funktionalität**				
	Gleichheit der Ausgaben	**Vollständigkeit der Übersetzung**	**Richtigkeit**		
2	Die Ausgaben des Python-Codes und des COBOL-Codes sind identisch.	Der Python-Code übersetzt und implementiert alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert ohne Fehler und alle Funktionen können ohne Ausführungsfehler ausgeführt werden.		
1	Die Ausgaben des Python-Codes und des COBOL-Codes sind ähnlich, aber nicht identisch.	Der Python-Code übersetzt und implementiert die meisten, aber nicht alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert mit einigen Warnungen, oder es treten bei der Ausführung einige kleinere Fehler auf, die jedoch die grundsätzliche Funktionsfähigkeit nicht beeinträchtigen. Fehler lassen sich mit wenigen zusätzlichen Prompts beheben.		
0	Die Ausgaben des Python-Codes und des COBOL-Codes unterscheiden sich erheblich.	Der Python-Code lässt wesentliche Funktionen, Verfahren oder Operationen des ursprünglichen COBOL-Codes aus oder implementiert sie falsch.	Der Python-Code kompiliert nicht, oder es treten bei der Ausführung schwere Fehler auf, die die Funktionsfähigkeit beeinträchtigen. Fehler lassen sich kaum beheben		
				Total Punkte	6
				Erreichte Punkte	0
				Erreichte Punkte in Prozent	0
	Optimalität				
	Grösse des Programms	**Redundanz**	**Einhaltung von Best Practices**		
2	Der Python-Code ist effizient geschrieben und so gut wie möglich komprimiert. Es gibt eine angemessene Anzahl von Leerzeilen zur Strukturierung.	Der Python-Code vermeidet redundante Operationen oder Wiederholungen. Code-Wiederverwendung wird maximiert.	Der Python-Code hält sich an allgemein anerkannte Best Practices für Python-Programmierung, einschliesslich Einrückung, Benennungskonventionen, Verwendung von Funktionen und Klassen usw.		
1	Der Python-Code ist im Allgemeinen effizient geschrieben, es gibt jedoch Möglichkeiten zur Optimierung.	Der Python-Code enthält einige redundante Operationen oder Wiederholungen, die reduziert werden könnten.	Der Python-Code hält sich grösstenteils an Best Practices, es gibt jedoch einige Verstösse.		
0	Der Python-Code ist ineffizient und zu lang. Es gibt zu viele Leerzeilen.	Der Python-Code ist voller redundanter Operationen oder Wiederholungen.	Der Python-Code verstösst häufig gegen Best Practices.		
				Total	6
				Erreichte Punkte	1
				Erreichte Punkte in Prozent	16.66667
	Verständlichkeit				
	Lesbarkeit	**Variablen und Funktionen**	**Kommentare**		
2	Der Python-Code ist einfach zu lesen und zu verstehen.	Variablen und Funktionen haben klare, beschreibende Namen, die ihrem Zweck im Code entsprechen.	Der Code ist gut kommentiert, und die Kommentare tragen effektiv zum Verständnis des Codes bei.		
1	Der Python-Code ist verständlich, enthält aber unklare Abschnitte.	Die meisten Variablen und Funktionen haben klare, beschreibende Namen, aber einige könnten verbessert werden.	Der Code hat einige nützliche Kommentare, aber es gibt Bereiche, die mehr Kommentierung erfordern oder wo die vorhandenen Kommentare nicht klar sind.		
0	Der Python-Code ist schwer zu lesen und zu verstehen.	Variablen und Funktionen haben unklare oder nicht beschreibende Namen.	Der Code hat wenige oder keine hilfreichen Kommentare.		
				Total Punkte	6
				Erreichte Punkte	0
				Erreichte Punkte in Prozent	0

Kommentar:
- 50% der Ausgaben sind fehlerhaft, weil die generierte Zufallszahl grösser als 100 ist. Dieser Fehler wird in den Zeilen 24 verursacht. W_tip wird im Array ein Wert zugewiesen, der im Index nicht verfügbar ist. -> IndexError: list index out of range
- Zeilen 19 bis 22 könnten zusammengefasst werden in w_j = random.randint(1, 100)
- ChatGPT hat Logikfehler bei übersetzung von COBOL in Python Code und übersetzt die For-Loops falsch
- Im Pythoncode wird w_random_tip (Zahl 1-200) generiert, die darauffolgenden 3 Zeilen Code zählen von 0 bis w_random_tip
- Keine Kommentare enthalten, Variablen und Funktionsnamen 1:1 übernommen, sagen aber nichts aus
- Code könnte in wenigen Zeilen korrekt geschrieben werden.

Kriterien zur Bewertung von ChatGPT (mit Anweisung)					
Punkte	**Funktionalität**				
	Gleichheit der Ausgaben	**Vollständigkeit der Übersetzung**	**Richtigkeit**		
2	Die Ausgaben des Python-Codes und des COBOL-Codes sind identisch.	Der Python-Code übersetzt und implementiert alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert ohne Fehler und alle Funktionen können ohne Ausführungsfehler ausgeführt werden.		
1	Die Ausgaben des Python-Codes und des COBOL-Codes sind ähnlich, aber nicht identisch.	Der Python-Code übersetzt und implementiert die meisten, aber nicht alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert mit einigen Warnungen, oder es treten bei der Ausführung einige kleinere Fehler auf, die jedoch die grundsätzliche Funktionsfähigkeit nicht beeinträchtigen. Fehler lassen sich mit wenigen zusätzlichen Prompts beheben.		
0	Die Ausgaben des Python-Codes und des COBOL-Codes unterscheiden sich erheblich.	Der Python-Code lässt wesentliche Funktionen, Verfahren oder Operationen des ursprünglichen COBOL-Codes aus oder implementiert sie falsch.	Der Python-Code kompiliert nicht, oder es treten bei der Ausführung schwere Fehler auf, die die Funktionsfähigkeit beeinträchtigen. Fehler lassen sich kaum beheben		
				Total Punkte	6
				Erreichte Punkte	6
				Erreichte Punkte in Prozent	100
	Optimalität				
	Grösse des Programms	**Redundanz**	**Einhaltung von Best Practices**		
2	Der Python-Code ist effizient geschrieben und so gut wie möglich komprimiert. Es gibt eine angemessene Anzahl von Leerzeilen zur Strukturierung.	Der Python-Code vermeidet redundante Operationen oder Wiederholungen. Code-Wiederverwendung wird maximiert.	Der Python-Code hält sich an allgemein anerkannte Best Practices für Python-Programmierung, einschliesslich Einrückung, Benennungskonventionen, Verwendung von Funktionen und Klassen usw.		
1	Der Python-Code ist im Allgemeinen effizient geschrieben, es gibt jedoch Möglichkeiten zur Optimierung.	Der Python-Code enthält einige redundante Operationen oder Wiederholungen, die reduziert werden könnten.	Der Python-Code hält sich grösstenteils an Best Practices, es gibt jedoch einige Verstösse.		
0	Der Python-Code ist ineffizient und zu lang. Es gibt zu viele Leerzeilen.	Der Python-Code ist voller redundanter Operationen oder Wiederholungen.	Der Python-Code verstösst häufig gegen Best Practices.		
				Total	6
				Erreichte Punkte	1
				Erreichte Punkte in Prozent	16.66667
	Verständlichkeit				
	Lesbarkeit	**Variablen und Funktionen**	**Kommentare**		
2	Der Python-Code ist einfach zu lesen und zu verstehen.	Variablen und Funktionen haben klare, beschreibende Namen, die ihrem Zweck im Code entsprechen.	Der Code ist gut kommentiert, und die Kommentare tragen effektiv zum Verständnis des Codes bei.		
1	Der Python-Code ist verständlich, enthält aber unklare Abschnitte.	Die meisten Variablen und Funktionen haben klare, beschreibende Namen, aber einige könnten verbessert werden.	Der Code hat einige nützliche Kommentare, aber es gibt Bereiche, die mehr Kommentierung erfordern oder wo die vorhandenen Kommentare nicht klar sind.		
0	Der Python-Code ist schwer zu lesen und zu verstehen.	Variablen und Funktionen haben unklare oder nicht beschreibende Namen.	Der Code hat wenige oder keine hilfreichen Kommentare.		
				Total Punkte	6
				Erreichte Punkte	2
				Erreichte Punkte in Prozent	33.33333
Kommentar:					
- Enthält Zeilen die nicht gebraucht werden, da Zahlen bereits zuvor bekannt sind (Zeile 32, 33, 35 ...)					
- Gleicher Code könnte in 1 bis 2 Zeilen geschrieben werden (ohne Prints)					
- ChatGPT probiert Code aus COBOL 1:1 zu kopieren und verwendet keine für Python gangbaren Methoden. Intention hinter dem Code wurde von ChatGPT scheinbar nicht verstanden und daher nicht optimal umgesetzt.					
- Kommentare vorhanenden, jedoch wenig Mehrwert					
- Variablen und Funktionsnamen 1:1 übernommen, sagen aber nichts aus					
- Keine Fehler vorhanden beim kompilieren, gleiche Funktionalität wie COBOL Code					

Randomnumber

Kriterien zur Bewertung von ChatGPT (ohne Anweisung)					
Punkte	**Funktionalität**				
	Gleichheit der Ausgaben	**Vollständigkeit der Übersetzung**	**Richtigkeit**		
2	Die Ausgaben des Python-Codes und des COBOL-Codes sind identisch.	Der Python-Code übersetzt und implementiert alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert ohne Fehler und alle Funktionen können ohne Ausführungsfehler ausgeführt werden.		
1	Die Ausgaben des Python-Codes und des COBOL-Codes sind ähnlich, aber nicht identisch.	Der Python-Code übersetzt und implementiert die meisten, aber nicht alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert mit einigen Warnungen, oder es treten bei der Ausführung einige kleinere Fehler auf, die jedoch die grundsätzliche Funktionsfähigkeit nicht beeinträchtigen. Fehler lassen sich mit wenigen zusätzlichen Prompts beheben.		
0	Die Ausgaben des Python-Codes und des COBOL-Codes unterscheiden sich erheblich.	Der Python-Code lässt wesentliche Funktionen, Verfahren oder Operationen des ursprünglichen COBOL-Codes aus oder implementiert sie falsch.	Der Python-Code kompiliert nicht, oder es treten bei der Ausführung schwere Fehler auf, die die Funktionsfähigkeit beeinträchtigen. Fehler lassen sich kaum beheben		
				Total Punkte	6
				Erreichte Punkte	6
				Erreichte Punkte in Prozent	100
	Optimalität				
	Grösse des Programms	**Redundanz**	**Einhaltung von Best Practices**		
2	Der Python-Code ist effizient geschrieben und so gut wie möglich komprimiert. Es gibt eine angemessene Anzahl von Leerzeilen zur Strukturierung.	Der Python-Code vermeidet redundante Operationen oder Wiederholungen. Code-Wiederverwendung wird maximiert.	Der Python-Code hält sich an allgemein anerkannte Best Practices für Python-Programmierung, einschliesslich Einrückung, Benennungskonventionen, Verwendung von Funktionen und Klassen usw.		
1	Der Python-Code ist im Allgemeinen effizient geschrieben, es gibt jedoch Möglichkeiten zur Optimierung.	Der Python-Code enthält einige redundante Operationen oder Wiederholungen, die reduziert werden könnten.	Der Python-Code hält sich grösstenteils an Best Practices, es gibt jedoch einige Verstösse.		
0	Der Python-Code ist ineffizient und zu lang. Es gibt zu viele Leerzeilen.	Der Python-Code ist voller redundanter Operationen oder Wiederholungen.	Der Python-Code verstösst häufig gegen Best Practices.		
				Total	6
				Erreichte Punkte	5
				Erreichte Punkte in Prozent	83.33333
	Verständlichkeit				
	Lesbarkeit	**Variablen und Funktionen**	**Kommentare**		
2	Der Python-Code ist einfach zu lesen und zu verstehen.	Variablen und Funktionen haben klare, beschreibende Namen, die ihrem Zweck im Code entsprechen.	Der Code ist gut kommentiert, und die Kommentare tragen effektiv zum Verständnis des Codes bei.		
1	Der Python-Code ist verständlich, enthält aber unklare Abschnitte.	Die meisten Variablen und Funktionen haben klare, beschreibende Namen, aber einige könnten verbessert werden.	Der Code hat einige nützliche Kommentare, aber es gibt Bereiche, die mehr Kommentierung erfordern oder wo die vorhandenen Kommentare nicht klar sind.		
0	Der Python-Code ist schwer zu lesen und zu verstehen.	Variablen und Funktionen haben unklare oder nicht beschreibende Namen.	Der Code hat wenige oder keine hilfreichen Kommentare.		
				Total Punkte	6
				Erreichte Punkte	3
				Erreichte Punkte in Prozent	50
Kommentar:					
- Variablen und Funktionen könnten besser benannt werden. Main ist kein beschreibender Funktionsname.					
- Fehlende Main Methode					

Kriterien zur Bewertung von ChatGPT (mit Anweisung)					
Punkte	**Funktionalität**				
	Gleichheit der Ausgaben	**Vollständigkeit der Übersetzung**	**Richtigkeit**		
2	Die Ausgaben des Python-Codes und des COBOL-Codes sind identisch.	Der Python-Code übersetzt und implementiert alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert ohne Fehler und alle Funktionen können ohne Ausführungsfehler ausgeführt werden.		
1	Die Ausgaben des Python-Codes und des COBOL-Codes sind ähnlich, aber nicht identisch.	Der Python-Code übersetzt und implementiert die meisten, aber nicht alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert mit einigen Warnungen, oder es treten bei der Ausführung einige kleinere Fehler auf, die jedoch die grundsätzliche Funktionsfähigkeit nicht beeinträchtigen. Fehler lassen sich mit wenigen zusätzlichen Prompts beheben.		
0	Die Ausgaben des Python-Codes und des COBOL-Codes unterscheiden sich erheblich.	Der Python-Code lässt wesentliche Funktionen, Verfahren oder Operationen des ursprünglichen COBOL-Codes aus oder implementiert sie falsch.	Der Python-Code kompiliert nicht, oder es treten bei der Ausführung schwere Fehler auf, die die Funktionsfähigkeit beeinträchtigen. Fehler lassen sich kaum beheben		
				Total Punkte	6
				Erreichte Punkte	6
				Erreichte Punkte in Prozent	100
	Optimalität				
	Grösse des Programms	**Redundanz**	**Einhaltung von Best Practices**		
2	Der Python-Code ist effizient geschrieben und so gut wie möglich komprimiert. Es gibt eine angemessene Anzahl von Leerzeilen zur Strukturierung.	Der Python-Code vermeidet redundante Operationen oder Wiederholungen. Code-Wiederverwendung wird maximiert.	Der Python-Code hält sich an allgemein anerkannte Best Practices für Python-Programmierung, einschliesslich Einrückung, Benennungskonventionen, Verwendung von Funktionen und Klassen usw.		
1	Der Python-Code ist im Allgemeinen effizient geschrieben, es gibt jedoch Möglichkeiten zur Optimierung.	Der Python-Code enthält einige redundante Operationen oder Wiederholungen, die reduziert werden könnten.	Der Python-Code hält sich grösstenteils an Best Practices, es gibt jedoch einige Verstösse.		
0	Der Python-Code ist ineffizient und zu lang. Es gibt zu viele Leerzeilen.	Der Python-Code ist voller redundanter Operationen oder Wiederholungen.	Der Python-Code verstösst häufig gegen Best Practices.		
				Total	6
				Erreichte Punkte	4
				Erreichte Punkte in Prozent	66.66667
	Verständlichkeit				
	Lesbarkeit	**Variablen und Funktionen**	**Kommentare**		
2	Der Python-Code ist einfach zu lesen und zu verstehen.	Variablen und Funktionen haben klare, beschreibende Namen, die ihrem Zweck im Code entsprechen.	Der Code ist gut kommentiert, und die Kommentare tragen effektiv zum Verständnis des Codes bei.		
1	Der Python-Code ist verständlich, enthält aber unklare Abschnitte.	Die meisten Variablen und Funktionen haben klare, beschreibende Namen, aber einige könnten verbessert werden.	Der Code hat einige nützliche Kommentare, aber es gibt Bereiche, die mehr Kommentierung erfordern oder wo die vorhandenen Kommentare nicht klar sind.		
0	Der Python-Code ist schwer zu lesen und zu verstehen.	Variablen und Funktionen haben unklare oder nicht beschreibende Namen.	Der Code hat wenige oder keine hilfreichen Kommentare.		
				Total Punkte	6
				Erreichte Punkte	5
				Erreichte Punkte in Prozent	83.33333
Kommentar:					
- Variablen und Funktionen könnten besser benannt werden. Main ist kein beschreibender Funktionsname.					
- w_result = 0 ist nicht nötig vorab festzulegen					

Randomsort

Kriterien zur Bewertung von ChatGPT (ohne Anweisung)					
Punkte		**Funktionalität**			
	Gleichheit der Ausgaben	**Vollständigkeit der Übersetzung**	**Richtigkeit**		
2	Die Ausgaben des Python-Codes und des COBOL-Codes sind identisch.	Der Python-Code übersetzt und implementiert alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert ohne Fehler und alle Funktionen können ohne Ausführungsfehler ausgeführt werden.		
1	Die Ausgaben des Python-Codes und des COBOL-Codes sind ähnlich, aber nicht identisch.	Der Python-Code übersetzt und implementiert die meisten, aber nicht alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert mit einigen Warnungen, oder es treten bei der Ausführung einige kleinere Fehler auf, die jedoch die grundsätzliche Funktionsfähigkeit nicht beeinträchtigen. Fehler lassen sich mit wenigen zusätzlichen Prompts beheben.		
0	Die Ausgaben des Python-Codes und des COBOL-Codes unterscheiden sich erheblich.	Der Python-Code lässt wesentliche Funktionen, Verfahren oder Operationen des ursprünglichen COBOL-Codes aus oder implementiert sie falsch.	Der Python-Code kompiliert nicht, oder es treten bei der Ausführung schwere Fehler auf, die die Funktionsfähigkeit beeinträchtigen. Fehler lassen sich kaum beheben		
				Total Punkte	6
				Erreichte Punkte	6
				Erreichte Punkte in Prozent	100
		Optimalität			
	Grösse des Programms	**Redundanz**	**Einhaltung von Best Practices**		
2	Der Python-Code ist effizient geschrieben und so gut wie möglich komprimiert. Es gibt eine angemessene Anzahl von Leerzeilen zur Strukturierung.	Der Python-Code vermeidet redundante Operationen oder Wiederholungen. Code-Wiederverwendung wird maximiert.	Der Python-Code hält sich an allgemein anerkannte Best Practices für Python-Programmierung, einschliesslich Einrückung, Benennungskonventionen, Verwendung von Funktionen und Klassen usw.		
1	Der Python-Code ist im Allgemeinen effizient geschrieben, es gibt jedoch Möglichkeiten zur Optimierung.	Der Python-Code enthält einige redundante Operationen oder Wiederholungen, die reduziert werden könnten.	Der Python-Code hält sich grösstenteils an Best Practices, es gibt jedoch einige Verstösse.		
0	Der Python-Code ist ineffizient und zu lang. Es gibt zu viele Leerzeilen.	Der Python-Code ist voller redundanter Operationen oder Wiederholungen.	Der Python-Code verstösst häufig gegen Best Practices.		
				Total	6
				Erreichte Punkte	5
				Erreichte Punkte in Prozent	83.33333
		Verständlichkeit			
	Lesbarkeit	**Variablen und Funktionen**	**Kommentare**		
2	Der Python-Code ist einfach zu lesen und zu verstehen.	Variablen und Funktionen haben klare, beschreibende Namen, die ihrem Zweck im Code entsprechen.	Der Code ist gut kommentiert, und die Kommentare tragen effektiv zum Verständnis des Codes bei.		
1	Der Python-Code ist verständlich, enthält aber unklare Abschnitte.	Die meisten Variablen und Funktionen haben klare, beschreibende Namen, aber einige könnten verbessert werden.	Der Code hat einige nützliche Kommentare, aber es gibt Bereiche, die mehr Kommentierung erfordern oder wo die vorhandenen Kommentare nicht klar sind.		
0	Der Python-Code ist schwer zu lesen und zu verstehen.	Variablen und Funktionen haben unklare oder nicht beschreibende Namen.	Der Code hat wenige oder keine hilfreichen Kommentare.		
				Total Punkte	6
				Erreichte Punkte	4
				Erreichte Punkte in Prozent	66.66667
Kommentar:					
- Verwendet keine Main Methode					
- Funktions- und Variablenbenennung 1:1 kopiert, wenig aussagekräftig.					

Kriterien zur Bewertung von ChatGPT (mit Anweisung)					
Punkte	**Funktionalität**				
	Gleichheit der Ausgaben	**Vollständigkeit der Übersetzung**	**Richtigkeit**		
2	Die Ausgaben des Python-Codes und des COBOL-Codes sind identisch.	Der Python-Code übersetzt und implementiert alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert ohne Fehler und alle Funktionen können ohne Ausführungsfehler ausgeführt werden.		
1	Die Ausgaben des Python-Codes und des COBOL-Codes sind ähnlich, aber nicht identisch.	Der Python-Code übersetzt und implementiert die meisten, aber nicht alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert mit einigen Warnungen, oder es treten bei der Ausführung einige kleinere Fehler auf, die jedoch die grundsätzliche Funktionsfähigkeit nicht beeinträchtigen. Fehler lassen sich mit wenigen zusätzlichen Prompts beheben.		
0	Die Ausgaben des Python-Codes und des COBOL-Codes unterscheiden sich erheblich.	Der Python-Code lässt wesentliche Funktionen, Verfahren oder Operationen des ursprünglichen COBOL-Codes aus oder implementiert sie falsch.	Der Python-Code kompiliert nicht, oder es treten bei der Ausführung schwere Fehler auf, die die Funktionsfähigkeit beeinträchtigen. Fehler lassen sich kaum beheben		
				Total Punkte	6
				Erreichte Punkte	6
				Erreichte Punkte in Prozent	100
	Optimalität				
	Grösse des Programms	**Redundanz**	**Einhaltung von Best Practices**		
2	Der Python-Code ist effizient geschrieben und so gut wie möglich komprimiert. Es gibt eine angemessene Anzahl von Leerzeilen zur Strukturierung.	Der Python-Code vermeidet redundante Operationen oder Wiederholungen. Code-Wiederverwendung wird maximiert.	Der Python-Code hält sich an allgemein anerkannte Best Practices für Python-Programmierung, einschliesslich Einrückung, Benennungskonventionen, Verwendung von Funktionen und Klassen usw.		
1	Der Python-Code ist im Allgemeinen effizient geschrieben, es gibt jedoch Möglichkeiten zur Optimierung.	Der Python-Code enthält einige redundante Operationen oder Wiederholungen, die reduziert werden könnten.	Der Python-Code hält sich grösstenteils an Best Practices, es gibt jedoch einige Verstösse.		
0	Der Python-Code ist ineffizient und zu lang. Es gibt zu viele Leerzeilen.	Der Python-Code ist voller redundanter Operationen oder Wiederholungen.	Der Python-Code verstösst häufig gegen Best Practices.		
				Total	6
				Erreichte Punkte	6
				Erreichte Punkte in Prozent	100
	Verständlichkeit				
	Lesbarkeit	**Variablen und Funktionen**	**Kommentare**		
2	Der Python-Code ist einfach zu lesen und zu verstehen.	Variablen und Funktionen haben klare, beschreibende Namen, die ihrem Zweck im Code entsprechen.	Der Code ist gut kommentiert, und die Kommentare tragen effektiv zum Verständnis des Codes bei.		
1	Der Python-Code ist verständlich, enthält aber unklare Abschnitte.	Die meisten Variablen und Funktionen haben klare, beschreibende Namen, aber einige könnten verbessert werden.	Der Code hat einige nützliche Kommentare, aber es gibt Bereiche, die mehr Kommentierung erfordern oder wo die vorhandenen Kommentare nicht klar sind.		
0	Der Python-Code ist schwer zu lesen und zu verstehen.	Variablen und Funktionen haben unklare oder nicht beschreibende Namen.	Der Code hat wenige oder keine hilfreichen Kommentare.		
				Total Punkte	6
				Erreichte Punkte	5
				Erreichte Punkte in Prozent	83.33333
Kommentar:					
- Funktions- und Variablenbenennung 1:1 kopiert, wenig aussagekräftig.					

Simplecalculator

Kriterien zur Bewertung von ChatGPT (ohne Anweisung)					
Punkte	**Funktionalität**				
	Gleichheit der Ausgaben	**Vollständigkeit der Übersetzung**	**Richtigkeit**		
2	Die Ausgaben des Python-Codes und des COBOL-Codes sind identisch.	Der Python-Code übersetzt und implementiert alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert ohne Fehler und alle Funktionen können ohne Ausführungsfehler ausgeführt werden.		
1	Die Ausgaben des Python-Codes und des COBOL-Codes sind ähnlich, aber nicht identisch.	Der Python-Code übersetzt und implementiert die meisten, aber nicht alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert mit einigen Warnungen, oder es treten bei der Ausführung einige kleinere Fehler auf, die jedoch die grundsätzliche Funktionsfähigkeit nicht beeinträchtigen. Fehler lassen sich mit wenigen zusätzlichen Prompts beheben.		
0	Die Ausgaben des Python-Codes und des COBOL-Codes unterscheiden sich erheblich.	Der Python-Code lässt wesentliche Funktionen, Verfahren oder Operationen des ursprünglichen COBOL-Codes aus oder implementiert sie falsch.	Der Python-Code kompiliert nicht, oder es treten bei der Ausführung schwere Fehler auf, die die Funktionsfähigkeit beeinträchtigen. Fehler lassen sich kaum beheben		
				Total Punkte	6
				Erreichte Punkte	6
				Erreichte Punkte in Prozent	100
	Optimalität				
	Grösse des Programms	**Redundanz**	**Einhaltung von Best Practices**		
2	Der Python-Code ist effizient geschrieben und so gut wie möglich komprimiert. Es gibt eine angemessene Anzahl von Leerzeilen zur Strukturierung.	Der Python-Code vermeidet redundante Operationen oder Wiederholungen. Code-Wiederverwendung wird maximiert.	Der Python-Code hält sich an allgemein anerkannte Best Practices für Python-Programmierung, einschliesslich Einrückung, Benennungskonventionen, Verwendung von Funktionen und Klassen usw.		
1	Der Python-Code ist im Allgemeinen effizient geschrieben, es gibt jedoch Möglichkeiten zur Optimierung.	Der Python-Code enthält einige redundante Operationen oder Wiederholungen, die reduziert werden könnten.	Der Python-Code hält sich grösstenteils an Best Practices, es gibt jedoch einige Verstösse.		
0	Der Python-Code ist ineffizient und zu lang. Es gibt zu viele Leerzeilen.	Der Python-Code ist voller redundanter Operationen oder Wiederholungen.	Der Python-Code verstösst häufig gegen Best Practices.		
				Total Punkte	6
				Erreichte Punkte	5
				Erreichte Punkte in Prozent	83.33333
	Verständlichkeit				
	Lesbarkeit	**Variablen und Funktionen**	**Kommentare**		
2	Der Python-Code ist einfach zu lesen und zu	Variablen und Funktionen haben klare,	Der Code ist gut kommentiert, und die		
1	Der Python-Code ist verständlich, enthält aber unklare Abschnitte.	Die meisten Variablen und Funktionen haben klare, beschreibende Namen, aber einige könnten verbessert werden.	Der Code hat einige nützliche Kommentare, aber es gibt Bereiche, die mehr Kommentierung erfordern oder wo die vorhandenen Kommentare nicht klar sind.		
0	Der Python-Code ist schwer zu lesen und zu verstehen.	Variablen und Funktionen haben unklare oder nicht beschreibende Namen.	Der Code hat wenige oder keine hilfreichen Kommentare.		
				Total Punkte	6
				Erreichte Punkte	3
				Erreichte Punkte in Prozent	50

Kommentar:
- Enthält Zeilen die nix machen, verwendet werden oder nutzlos sind. Z.B. Zeile 39, und Variable estado (Zeile 5)
- Variablen hätten klarer benannt werden können. Z.B. first_number anstatt n1
- Keine Kommentare Vorhanden
- Verbessert Funktionalität im Vergleich zum COBOL Code deutlich

Kriterien zur Bewertung von ChatGPT (mit Anweisung)					
Punkte	**Funktionalität**				
	Gleichheit der Ausgaben	**Vollständigkeit der Übersetzung**	**Richtigkeit**		
2	Die Ausgaben des Python-Codes und des COBOL-Codes sind identisch.	Der Python-Code übersetzt und implementiert alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert ohne Fehler und alle Funktionen können ohne Ausführungsfehler ausgeführt werden.		
1	Die Ausgaben des Python-Codes und des COBOL-Codes sind ähnlich, aber nicht identisch.	Der Python-Code übersetzt und implementiert die meisten, aber nicht alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert mit einigen Warnungen, oder es treten bei der Ausführung einige kleinere Fehler auf, die jedoch die grundsätzliche Funktionsfähigkeit nicht beeinträchtigen. Fehler lassen sich mit wenigen zusätzlichen Prompts beheben.		
0	Die Ausgaben des Python-Codes und des COBOL-Codes unterscheiden sich erheblich.	Der Python-Code lässt wesentliche Funktionen, Verfahren oder Operationen des ursprünglichen COBOL-Codes aus oder implementiert sie falsch.	Der Python-Code kompiliert nicht, oder es treten bei der Ausführung schwere Fehler auf, die die Funktionsfähigkeit beeinträchtigen. Fehler lassen sich kaum beheben		
				Total Punkte	6
				Erreichte Punkte	5
				Erreichte Punkte in Prozent	83.33333
Optimalität					
	Grösse des Programms	**Redundanz**	**Einhaltung von Best Practices**		
2	Der Python-Code ist effizient geschrieben und so gut wie möglich komprimiert. Es gibt eine angemessene Anzahl von Leerzeilen zur Strukturierung.	Der Python-Code vermeidet redundante Operationen oder Wiederholungen. Code-Wiederverwendung wird maximiert.	Der Python-Code hält sich an allgemein anerkannte Best Practices für Python-Programmierung, einschliesslich Einrückung, Benennungskonventionen, Verwendung von Funktionen und Klassen usw.		
1	Der Python-Code ist im Allgemeinen effizient geschrieben, es gibt jedoch Möglichkeiten zur Optimierung.	Der Python-Code enthält einige redundante Operationen oder Wiederholungen, die reduziert werden könnten.	Der Python-Code hält sich grösstenteils an Best Practices, es gibt jedoch einige Verstösse.		
0	Der Python-Code ist ineffizient und zu lang. Es gibt zu viele Leerzeilen.	Der Python-Code ist voller redundanter Operationen oder Wiederholungen.	Der Python-Code verstösst häufig gegen Best Practices.		
				Total Punkte	6
				Erreichte Punkte	3
				Erreichte Punkte in Prozent	50
Verständlichkeit					
	Lesbarkeit	**Variablen und Funktionen**	**Kommentare**		
2	Der Python-Code ist einfach zu lesen und zu	Variablen und Funktionen haben klare,	Der Code ist gut kommentiert, und die		
1	Der Python-Code ist verständlich, enthält aber unklare Abschnitte.	Die meisten Variablen und Funktionen haben klare, beschreibende Namen, aber einige könnten verbessert werden.	Der Code hat einige nützliche Kommentare, aber es gibt Bereiche, die mehr Kommentierung erfordern oder wo die vorhandenen Kommentare nicht klar sind.		
0	Der Python-Code ist schwer zu lesen und zu verstehen.	Variablen und Funktionen haben unklare oder nicht beschreibende Namen.	Der Code hat wenige oder keine hilfreichen Kommentare.		
				Total Punkte	6
				Erreichte Punkte	2
				Erreichte Punkte in Prozent	33.33333
Kommentar:					
- Verschachtelte While Loops machen Code schwerer verständlich und blasen ihn auf					
- Variablen hätten klarer benannt werden können. Z.B. first_number anstatt n1, operator statt op					
- Nur ein Kommentar ohne zusätzlichen Wert					
- Probiert Error Handling einzubauen (Zeile 47-49) was von der Idee her gut ist, Invalid input wird aber nur ein Bruchteil einer Sekunde dem Nutzer angezeigt, was den Nutzen relativiert					
- Verbessert Funktionalität im Vergleich zum COBOL Code deutlich					

Stringsplit

Kriterien zur Bewertung von ChatGPT (ohne Anweisung)					
Punkte	**Funktionalität**				
	Gleichheit der Ausgaben	**Vollständigkeit der Übersetzung**	**Richtigkeit**		
2	Die Ausgaben des Python-Codes und des COBOL-Codes sind identisch.	Der Python-Code übersetzt und implementiert alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert ohne Fehler und alle Funktionen können ohne Ausführungsfehler ausgeführt werden.		
1	Die Ausgaben des Python-Codes und des COBOL-Codes sind ähnlich, aber nicht identisch.	Der Python-Code übersetzt und implementiert die meisten, aber nicht alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert mit einigen Warnungen, oder es treten bei der Ausführung einige kleinere Fehler auf, die jedoch die grundsätzliche Funktionsfähigkeit nicht beeinträchtigen. Fehler lassen sich mit wenigen zusätzlichen Prompts beheben.		
0	Die Ausgaben des Python-Codes und des COBOL-Codes unterscheiden sich erheblich.	Der Python-Code lässt wesentliche Funktionen, Verfahren oder Operationen des ursprünglichen COBOL-Codes aus oder implementiert sie falsch.	Der Python-Code kompiliert nicht, oder es treten bei der Ausführung schwere Fehler auf, die die Funktionsfähigkeit beeinträchtigen. Fehler lassen sich kaum beheben		
				Total Punkte	6
				Erreichte Punkte	6
				Erreichte Punkte in Prozent	100
	Optimalität				
	Grösse des Programms	**Redundanz**	**Einhaltung von Best Practices**		
2	Der Python-Code ist effizient geschrieben und so gut wie möglich komprimiert. Es gibt eine angemessene Anzahl von Leerzeilen zur Strukturierung.	Der Python-Code vermeidet redundante Operationen oder Wiederholungen. Code-Wiederverwendung wird maximiert.	Der Python-Code hält sich an allgemein anerkannte Best Practices für Python-Programmierung, einschliesslich Einrückung, Benennungskonventionen, Verwendung von Funktionen und Klassen usw.		
1	Der Python-Code ist im Allgemeinen effizient geschrieben, es gibt jedoch Möglichkeiten zur Optimierung.	Der Python-Code enthält einige redundante Operationen oder Wiederholungen, die reduziert werden könnten.	Der Python-Code hält sich grösstenteils an Best Practices, es gibt jedoch einige Verstösse.		
0	Der Python-Code ist ineffizient und zu lang. Es gibt zu viele Leerzeilen.	Der Python-Code ist voller redundanter Operationen oder Wiederholungen.	Der Python-Code verstösst häufig gegen Best Practices.		
				Total Punkte	6
				Erreichte Punkte	5
				Erreichte Punkte in Prozent	83.33333
	Verständlichkeit				
	Lesbarkeit	**Variablen und Funktionen**	**Kommentare**		
2	Der Python-Code ist einfach zu lesen und zu	Variablen und Funktionen haben klare,	Der Code ist gut kommentiert, und die		
1	Der Python-Code ist verständlich, enthält aber unklare Abschnitte.	Die meisten Variablen und Funktionen haben klare, beschreibende Namen, aber einige könnten verbessert werden.	Der Code hat einige nützliche Kommentare, aber es gibt Bereiche, die mehr Kommentierung erfordern oder wo die vorhandenen Kommentare nicht klar sind.		
0	Der Python-Code ist schwer zu lesen und zu verstehen.	Variablen und Funktionen haben unklare oder nicht beschreibende Namen.	Der Code hat wenige oder keine hilfreichen Kommentare.		
				Total Punkte	6
				Erreichte Punkte	3
				Erreichte Punkte in Prozent	50
Kommentar:					
- Funktion und Variablennamen 1:1 übernommen, könnten aussagekräftiger sein					
- Zeile 4 könnte in der Klammer sein range(len(w_string), damit auch längere Strings eingespeist werden könnten.					
- Enthält Main Methode					

Kriterien zur Bewertung von ChatGPT (mit Anweisung)					
Punkte	**Funktionalität**				
	Gleichheit der Ausgaben	**Vollständigkeit der Übersetzung**	**Richtigkeit**		
2	Die Ausgaben des Python-Codes und des COBOL-Codes sind identisch.	Der Python-Code übersetzt und implementiert alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert ohne Fehler und alle Funktionen können ohne Ausführungsfehler ausgeführt werden.		
1	Die Ausgaben des Python-Codes und des COBOL-Codes sind ähnlich, aber nicht identisch.	Der Python-Code übersetzt und implementiert die meisten, aber nicht alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert mit einigen Warnungen, oder es treten bei der Ausführung einige kleinere Fehler auf, die jedoch die grundsätzliche Funktionsfähigkeit nicht beeinträchtigen. Fehler lassen sich mit wenigen zusätzlichen Prompts beheben.		
0	Die Ausgaben des Python-Codes und des COBOL-Codes unterscheiden sich erheblich.	Der Python-Code lässt wesentliche Funktionen, Verfahren oder Operationen des ursprünglichen COBOL-Codes aus oder implementiert sie falsch.	Der Python-Code kompiliert nicht, oder es treten bei der Ausführung schwere Fehler auf, die die Funktionsfähigkeit beeinträchtigen. Fehler lassen sich kaum beheben		
				Total Punkte	6
				Erreichte Punkte	6
				Erreichte Punkte in Prozent	100
Optimalität					
	Grösse des Programms	**Redundanz**	**Einhaltung von Best Practices**		
2	Der Python-Code ist effizient geschrieben und so gut wie möglich komprimiert. Es gibt eine angemessene Anzahl von Leerzeilen zur Strukturierung.	Der Python-Code vermeidet redundante Operationen oder Wiederholungen. Code-Wiederverwendung wird maximiert.	Der Python-Code hält sich an allgemein anerkannte Best Practices für Python-Programmierung, einschliesslich Einrückung, Benennungskonventionen, Verwendung von Funktionen und Klassen usw.		
1	Der Python-Code ist im Allgemeinen effizient geschrieben, es gibt jedoch Möglichkeiten zur Optimierung.	Der Python-Code enthält einige redundante Operationen oder Wiederholungen, die reduziert werden könnten.	Der Python-Code hält sich grösstenteils an Best Practices, es gibt jedoch einige Verstösse.		
0	Der Python-Code ist ineffizient und zu lang. Es gibt zu viele Leerzeilen.	Der Python-Code ist voller redundanter Operationen oder Wiederholungen.	Der Python-Code verstösst häufig gegen Best Practices.		
				Total Punkte	6
				Erreichte Punkte	3
				Erreichte Punkte in Prozent	50
Verständlichkeit					
	Lesbarkeit	**Variablen und Funktionen**	**Kommentare**		
2	Der Python-Code ist einfach zu lesen und zu	Variablen und Funktionen haben klare,	Der Code ist gut kommentiert, und die		
1	Der Python-Code ist verständlich, enthält aber unklare Abschnitte.	Die meisten Variablen und Funktionen haben klare, beschreibende Namen, aber einige könnten verbessert werden.	Der Code hat einige nützliche Kommentare, aber es gibt Bereiche, die mehr Kommentierung erfordern oder wo die vorhandenen Kommentare nicht klar sind.		
0	Der Python-Code ist schwer zu lesen und zu verstehen.	Variablen und Funktionen haben unklare oder nicht beschreibende Namen.	Der Code hat wenige oder keine hilfreichen Kommentare.		
				Total Punkte	6
				Erreichte Punkte	4
				Erreichte Punkte in Prozent	66.66667
Kommentar:					
- Redundant, da zur String Länge +1 hinzugefügt und danach wieder abgezogen wird					
- Enthält Kommentare, sind aber eher als Rechtfertigung für "fehlerhaftes" Verhalten zu interpretieren					
- Keine Main Methode					

Trim

Kriterien zur Bewertung von ChatGPT (ohne Anweisung)					
Punkte		**Funktionalität**			
	Gleichheit der Ausgaben	**Vollständigkeit der Übersetzung**	**Richtigkeit**		
2	Die Ausgaben des Python-Codes und des COBOL-Codes sind identisch.	Der Python-Code übersetzt und implementiert alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert ohne Fehler und alle Funktionen können ohne Ausführungsfehler ausgeführt werden.		
1	Die Ausgaben des Python-Codes und des COBOL-Codes sind ähnlich, aber nicht identisch.	Der Python-Code übersetzt und implementiert die meisten, aber nicht alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert mit einigen Warnungen, oder es treten bei der Ausführung einige kleinere Fehler auf, die jedoch die grundsätzliche Funktionsfähigkeit nicht beeinträchtigen. Fehler lassen sich mit wenigen zusätzlichen Prompts beheben.		
0	Die Ausgaben des Python-Codes und des COBOL-Codes unterscheiden sich erheblich.	Der Python-Code lässt wesentliche Funktionen, Verfahren oder Operationen des ursprünglichen COBOL-Codes aus oder implementiert sie falsch.	Der Python-Code kompiliert nicht, oder es treten bei der Ausführung schwere Fehler auf, die die Funktionsfähigkeit beeinträchtigen. Fehler lassen sich kaum beheben		
				Total Punkte	6
				Erreichte Punkte	6
				Erreichte Punkte in Prozent	100
		Optimalität			
	Grösse des Programms	**Redundanz**	**Einhaltung von Best Practices**		
2	Der Python-Code ist effizient geschrieben und so gut wie möglich komprimiert. Es gibt eine angemessene Anzahl von Leerzeilen zur Strukturierung.	Der Python-Code vermeidet redundante Operationen oder Wiederholungen. Code-Wiederverwendung wird maximiert.	Der Python-Code hält sich an allgemein anerkannte Best Practices für Python-Programmierung, einschliesslich Einrückung, Benennungskonventionen, Verwendung von Funktionen und Klassen usw.		
1	Der Python-Code ist im Allgemeinen effizient geschrieben, es gibt jedoch Möglichkeiten zur Optimierung.	Der Python-Code enthält einige redundante Operationen oder Wiederholungen, die reduziert werden könnten.	Der Python-Code hält sich grösstenteils an Best Practices, es gibt jedoch einige Verstösse.		
0	Der Python-Code ist ineffizient und zu lang. Es gibt zu viele Leerzeilen.	Der Python-Code ist voller redundanter Operationen oder Wiederholungen.	Der Python-Code verstösst häufig gegen Best Practices.		
				Total Punkte	6
				Erreichte Punkte	0
				Erreichte Punkte in Prozent	0
		Verständlichkeit			
	Lesbarkeit	**Variablen und Funktionen**	**Kommentare**		
2	Der Python-Code ist einfach zu lesen und zu	Variablen und Funktionen haben klare,	Der Code ist gut kommentiert, und die		
1	Der Python-Code ist verständlich, enthält aber unklare Abschnitte.	Die meisten Variablen und Funktionen haben klare, beschreibende Namen, aber einige könnten verbessert werden.	Der Code hat einige nützliche Kommentare, aber es gibt Bereiche, die mehr Kommentierung erfordern oder wo die vorhandenen Kommentare nicht klar sind.		
0	Der Python-Code ist schwer zu lesen und zu verstehen.	Variablen und Funktionen haben unklare oder nicht beschreibende Namen.	Der Code hat wenige oder keine hilfreichen Kommentare.		
				Total Punkte	6
				Erreichte Punkte	0
				Erreichte Punkte in Prozent	0
Kommentar:					
- Funktionalität identisch mit COBOL Code					
- Loop verwenden statt immer dieselben Prints (z.B. Zeile 10 bis 28)					
- Variablenbenennungen unklar					
- Code dadurch viel zu lange und schwierig zu lesen					
- Keine Kommentare					

Kriterien zur Bewertung von ChatGPT (mit Anweisung)					
Punkte	**Funktionalität**				
	Gleichheit der Ausgaben	**Vollständigkeit der Übersetzung**	**Richtigkeit**		
2	Die Ausgaben des Python-Codes und des COBOL-Codes sind identisch.	Der Python-Code übersetzt und implementiert alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert ohne Fehler und alle Funktionen können ohne Ausführungsfehler ausgeführt werden.		
1	Die Ausgaben des Python-Codes und des COBOL-Codes sind ähnlich, aber nicht identisch.	Der Python-Code übersetzt und implementiert die meisten, aber nicht alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert mit einigen Warnungen, oder es treten bei der Ausführung einige kleinere Fehler auf, die jedoch die grundsätzliche Funktionsfähigkeit nicht beeinträchtigen. Fehler lassen sich mit wenigen zusätzlichen Prompts beheben.		
0	Die Ausgaben des Python-Codes und des COBOL-Codes unterscheiden sich erheblich.	Der Python-Code lässt wesentliche Funktionen, Verfahren oder Operationen des ursprünglichen COBOL-Codes aus oder implementiert sie falsch.	Der Python-Code kompiliert nicht, oder es treten bei der Ausführung schwere Fehler auf, die die Funktionsfähigkeit beeinträchtigen. Fehler lassen sich kaum beheben		
				Total Punkte	6
				Erreichte Punkte	6
				Erreichte Punkte in Prozent	100
Optimalität					
	Grösse des Programms	**Redundanz**	**Einhaltung von Best Practices**		
2	Der Python-Code ist effizient geschrieben und so gut wie möglich komprimiert. Es gibt eine angemessene Anzahl von Leerzeilen zur Strukturierung.	Der Python-Code vermeidet redundante Operationen oder Wiederholungen. Code-Wiederverwendung wird maximiert.	Der Python-Code hält sich an allgemein anerkannte Best Practices für Python-Programmierung, einschliesslich Einrückung, Benennungskonventionen, Verwendung von Funktionen und Klassen usw.		
1	Der Python-Code ist im Allgemeinen effizient geschrieben, es gibt jedoch Möglichkeiten zur Optimierung.	Der Python-Code enthält einige redundante Operationen oder Wiederholungen, die reduziert werden könnten.	Der Python-Code hält sich grösstenteils an Best Practices, es gibt jedoch einige Verstösse.		
0	Der Python-Code ist ineffizient und zu lang. Es gibt zu viele Leerzeilen.	Der Python-Code ist voller redundanter Operationen oder Wiederholungen.	Der Python-Code verstösst häufig gegen Best Practices.		
				Total Punkte	6
				Erreichte Punkte	0
				Erreichte Punkte in Prozent	0
Verständlichkeit					
	Lesbarkeit	**Variablen und Funktionen**	**Kommentare**		
2	Der Python-Code ist einfach zu lesen und zu	Variablen und Funktionen haben klare,	Der Code ist gut kommentiert, und die		
1	Der Python-Code ist verständlich, enthält aber unklare Abschnitte.	Die meisten Variablen und Funktionen haben klare, beschreibende Namen, aber einige könnten verbessert werden.	Der Code hat einige nützliche Kommentare, aber es gibt Bereiche, die mehr Kommentierung erfordern oder wo die vorhandenen Kommentare nicht klar sind.		
0	Der Python-Code ist schwer zu lesen und zu verstehen.	Variablen und Funktionen haben unklare oder nicht beschreibende Namen.	Der Code hat wenige oder keine hilfreichen Kommentare.		
				Total Punkte	6
				Erreichte Punkte	1
				Erreichte Punkte in Prozent	16.66667
Kommentar:					
- Funktionalität identisch mit COBOL Code					
- Loop verwenden statt immer dieselben Prints (z.B. Zeile 14 bis 31)					
- Variablenbenennungen unklar					
- Code dadurch viel zu lange und schwierig zu lesen					
- Kommentare stiften nur mässig Mehrwert					
- Verwendet keine Main Methode					

Whileloop

Kriterien zur Bewertung von ChatGPT (ohne Anweisung)					
Punkte	**Funktionalität**				
	Gleichheit der Ausgaben	**Vollständigkeit der Übersetzung**	**Richtigkeit**		
2	Die Ausgaben des Python-Codes und des COBOL-Codes sind identisch.	Der Python-Code übersetzt und implementiert alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert ohne Fehler und alle Funktionen können ohne Ausführungsfehler ausgeführt werden.		
1	Die Ausgaben des Python-Codes und des COBOL-Codes sind ähnlich, aber nicht identisch.	Der Python-Code übersetzt und implementiert die meisten, aber nicht alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert mit einigen Warnungen, oder es treten bei der Ausführung einige kleinere Fehler auf, die jedoch die grundsätzliche Funktionsfähigkeit nicht beeinträchtigen. Fehler lassen sich mit wenigen zusätzlichen Prompts beheben.		
0	Die Ausgaben des Python-Codes und des COBOL-Codes unterscheiden sich erheblich.	Der Python-Code lässt wesentliche Funktionen, Verfahren oder Operationen des ursprünglichen COBOL-Codes aus oder implementiert sie falsch.	Der Python-Code kompiliert nicht, oder es treten bei der Ausführung schwere Fehler auf, die die Funktionsfähigkeit beeinträchtigen. Fehler lassen sich kaum beheben		
				Total Punkte	6
				Erreichte Punkte	5
				Erreichte Punkte in Prozent	83.33333
	Optimalität				
	Grösse des Programms	**Redundanz**	**Einhaltung von Best Practices**		
2	Der Python-Code ist effizient geschrieben und so gut wie möglich komprimiert. Es gibt eine angemessene Anzahl von Leerzeilen zur Strukturierung.	Der Python-Code vermeidet redundante Operationen oder Wiederholungen. Code-Wiederverwendung wird maximiert.	Der Python-Code hält sich an allgemein anerkannte Best Practices für Python-Programmierung, einschliesslich Einrückung, Benennungskonventionen, Verwendung von Funktionen und Klassen usw.		
1	Der Python-Code ist im Allgemeinen effizient geschrieben, es gibt jedoch Möglichkeiten zur Optimierung.	Der Python-Code enthält einige redundante Operationen oder Wiederholungen, die reduziert werden könnten.	Der Python-Code hält sich grösstenteils an Best Practices, es gibt jedoch einige Verstösse.		
0	Der Python-Code ist ineffizient und zu lang. Es gibt zu viele Leerzeilen.	Der Python-Code ist voller redundanter Operationen oder Wiederholungen.	Der Python-Code verstösst häufig gegen Best Practices.		
				Total	6
				Erreichte Punkte	6
				Erreichte Punkte in Prozent	100
	Verständlichkeit				
	Lesbarkeit	**Variablen und Funktionen**	**Kommentare**		
2	Der Python-Code ist einfach zu lesen und zu verstehen.	Variablen und Funktionen haben klare, beschreibende Namen, die ihrem Zweck im Code entsprechen.	Der Code ist gut kommentiert, und die Kommentare tragen effektiv zum Verständnis des Codes bei.		
1	Der Python-Code ist verständlich, enthält aber unklare Abschnitte.	Die meisten Variablen und Funktionen haben klare, beschreibende Namen, aber einige könnten verbessert werden.	Der Code hat einige nützliche Kommentare, aber es gibt Bereiche, die mehr Kommentierung erfordern oder wo die vorhandenen Kommentare nicht klar sind.		
0	Der Python-Code ist schwer zu lesen und zu verstehen.	Variablen und Funktionen haben unklare oder nicht beschreibende Namen.	Der Code hat wenige oder keine hilfreichen Kommentare.		
				Total Punkte	6
				Erreichte Punkte	3
				Erreichte Punkte in Prozent	50
Kommentar:					
- Keine Kommentare					
- Ausgabe enthält auch 0, im COBOL Code wird nur bis 1 runtergezählt					
- Main Funktion Namensbenennung könnnte aussagekräftiger sein, dasselbe für w_i					

Kriterien zur Bewertung von ChatGPT (mit Anweisung)					
Punkte	**Funktionalität**				
	Gleichheit der Ausgaben	**Vollständigkeit der Übersetzung**	**Richtigkeit**		
2	Die Ausgaben des Python-Codes und des COBOL-Codes sind identisch.	Der Python-Code übersetzt und implementiert alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert ohne Fehler und alle Funktionen können ohne Ausführungsfehler ausgeführt werden.		
1	Die Ausgaben des Python-Codes und des COBOL-Codes sind ähnlich, aber nicht identisch.	Der Python-Code übersetzt und implementiert die meisten, aber nicht alle Funktionen, Verfahren und Operationen des ursprünglichen COBOL-Codes korrekt.	Der Python-Code kompiliert mit einigen Warnungen, oder es treten bei der Ausführung einige kleinere Fehler auf, die jedoch die grundsätzliche Funktionsfähigkeit nicht beeinträchtigen. Fehler lassen sich mit wenigen zusätzlichen Prompts beheben.		
0	Die Ausgaben des Python-Codes und des COBOL-Codes unterscheiden sich erheblich.	Der Python-Code lässt wesentliche Funktionen, Verfahren oder Operationen des ursprünglichen COBOL-Codes aus oder implementiert sie falsch.	Der Python-Code kompiliert nicht, oder es treten bei der Ausführung schwere Fehler auf, die die Funktionsfähigkeit beeinträchtigen. Fehler lassen sich kaum beheben		
				Total Punkte	6
				Erreichte Punkte	5
				Erreichte Punkte in Prozent	83.33333
Optimalität					
	Grösse des Programms	**Redundanz**	**Einhaltung von Best Practices**		
2	Der Python-Code ist effizient geschrieben und so gut wie möglich komprimiert. Es gibt eine angemessene Anzahl von Leerzeilen zur Strukturierung.	Der Python-Code vermeidet redundante Operationen oder Wiederholungen. Code-Wiederverwendung wird maximiert.	Der Python-Code hält sich an allgemein anerkannte Best Practices für Python-Programmierung, einschliesslich Einrückung, Benennungskonventionen, Verwendung von Funktionen und Klassen usw.		
1	Der Python-Code ist im Allgemeinen effizient geschrieben, es gibt jedoch Möglichkeiten zur Optimierung.	Der Python-Code enthält einige redundante Operationen oder Wiederholungen, die reduziert werden könnten.	Der Python-Code hält sich grösstenteils an Best Practices, es gibt jedoch einige Verstösse.		
0	Der Python-Code ist ineffizient und zu lang. Es gibt zu viele Leerzeilen.	Der Python-Code ist voller redundanter Operationen oder Wiederholungen.	Der Python-Code verstösst häufig gegen Best Practices.		
				Total	6
				Erreichte Punkte	6
				Erreichte Punkte in Prozent	100
Verständlichkeit					
	Lesbarkeit	**Variablen und Funktionen**	**Kommentare**		
2	Der Python-Code ist einfach zu lesen und zu verstehen.	Variablen und Funktionen haben klare, beschreibende Namen, die ihrem Zweck im Code entsprechen.	Der Code ist gut kommentiert, und die Kommentare tragen effektiv zum Verständnis des Codes bei.		
1	Der Python-Code ist verständlich, enthält aber unklare Abschnitte.	Die meisten Variablen und Funktionen haben klare, beschreibende Namen, aber einige könnten verbessert werden.	Der Code hat einige nützliche Kommentare, aber es gibt Bereiche, die mehr Kommentierung erfordern oder wo die vorhandenen Kommentare nicht klar sind.		
0	Der Python-Code ist schwer zu lesen und zu verstehen.	Variablen und Funktionen haben unklare oder nicht beschreibende Namen.	Der Code hat wenige oder keine hilfreichen Kommentare.		
				Total Punkte	6
				Erreichte Punkte	3
				Erreichte Punkte in Prozent	50
Kommentar:					
- Keine Kommentare					
- Ausgabe enthält auch 0, im COBOL Code wird nur bis 1 runtergezählt					
- Main Funktion Namensbenennung könnnte aussagekräftiger sein, dasselbe für w_i					
- Identischer Code wie mit einer Leerzeile weniger.					

Bisher erschienene Schriften

Ergebnisse von Forschungsprojekten erscheinen jeweils in Form von Arbeitsberichten in Reihen. Sonstige Publikationen erscheinen in Form von alleinstehenden Schriften.

Derzeit gibt es in den Churer Schriften zur Informationswissenschaft folgende Reihen:
Reihe Berufsmarktforschung

Weitere Publikationen

Churer Schriften zur Informationswissenschaft – Schrift 150
Herausgegeben von Wolfgang Semar
Nicole Fässler
User Adoption bei der Einführung einer Kollaborations- und Kommunikationssoft-ware im Modern Workplace Umfeld
Chur, 2022
ISSN 1660-945X

Churer Schriften zur Informationswissenschaft – Schrift 151
Herausgegeben von Wolfgang Semar
Marina Inglin
Re- und Upskilling-Empfehlung
Kriterien für die automatische Auswahl von Re- und Upskilling-Angeboten
Chur, 2022
ISSN 1660-945X

Churer Schriften zur Informationswissenschaft – Schrift 152
Herausgegeben von Wolfgang Semar
Lisa Heller
Zur Genese eines nationalen Bibliothekspro-jekts: Swiss Library Service Platform (SLSP)
Chur, 2022
ISSN 1660-945X

Churer Schriften zur Informationswissenschaft – Schrift 153
Herausgegeben von Wolfgang Semar
Antonin Friberg
Die Effektivität von Social Norms Nudging in der Customer Journey
Chur, 2022
ISSN 1660-945X

Churer Schriften zur Informationswissenschaft – Schrift 154
Herausgegeben von Wolfgang Semar
Curdin Marxer
«Drug Repurposing»
Wie können unstrukturierte Textdaten für die Ermitt-lung neuer «Drug Repurposing» Kandidaten nutzbar gemacht werden und wie können sie Datenbanken ergänzen?
Chur, 2022
ISSN 1660-945X

Churer Schriften zur Informationswissenschaft – Schrift 155
Herausgegeben von Wolfgang Semar
Samir Limani
Sicht der administrativen Mitarbeitenden von Bündner Spitälern und Kliniken auf den Digitalisierungsstand ihres Unternehmens
Chur, 2022
ISSN 1660-945X

Churer Schriften zur Informationswissenschaft – Schrift 156
Herausgegeben von Wolfgang Semar
Marina Lea Schürmann
Deep Learning für Part-of-Speech-Tagging
Vergleich eines auf Transformers basierenden POS-Taggers mit bestehenden Modellen
Chur, 2023
ISSN 1660-945X

Churer Schriften zur Informationswissenschaft – Schrift 157
Herausgegeben von Wolfgang Semar
Yannick Mireille Kaufmann
Einsatz von Unternehmenswikis als Wissens-management-Tool in einer Netzwerkorganisation
Evaluationsstudie zu «wikimia», eine Wissensdaten-bank in der schweizerischen Berufs-, Studien- und Laufbahnberatung Masterthesis 2022
Chur, 2023
ISSN 1660-945X

Churer Schriften zur Informationswissenschaft – Schrift 158
Herausgegeben von Wolfgang Semar
Franco Malacrida
Standortfindung von Schweizer Start-ups
Welche Standortfaktoren sind für Schweizer Start-ups am wichtigsten?
Chur, 2023
ISSN 1660-945X

Churer Schriften zur Informationswissenschaft – Schrift 159
Herausgegeben von Wolfgang Semar
Josip Spec
From ISAD(G) to Records in Contexts – A new era
Chur, 2023
ISSN 1660-945X

Churer Schriften zur Informationswissenschaft – Schrift 160
Herausgegeben von Wolfgang Semar
Loris Haller
Gemeinwohl fördern als Geschäftsmodell
Kriterien für die Entwicklung eines Frameworks für gemein-wohlorientierte Geschäftsmodelle
Chur, 2023
ISSN 1660-945X

Churer Schriften zur Informationswissenschaft – Schrift 161
Herausgegeben von Wolfgang Semar
Céline Graf
«Ghostbusters Münstergasse»
Vermittlung von regionalen Onlineressourcen und Recherchekompetenzen mit einem digitalen Educational Escape Room an der Bibliothek Münstergasse der Universitätsbibliothek Bern
Chur, 2023
ISSN 1660-945X

Churer Schriften zur Informationswissenschaft – Schrift 162
Herausgegeben von Wolfgang Semar
Mahmoud Hemila
Qualitätsanalyse von inhaltsbasierten Empfehlungssystemen für Journals
Chur, 2023
ISSN 1660-945X

Churer Schriften zur Informationswissenschaft – Schrift 163
Herausgegeben von Wolfgang Semar
Nicolas Brauchli
Inwiefern unterscheiden sich die Online-Plattformen der Legacy-Medien von den Digital Born Plattformen in der Deutschschweizer Medienlandschaft?
Chur, 2023
ISSN 1660-945X

Churer Schriften zur Informationswissenschaft – Schrift 164
Herausgegeben von Wolfgang Semar
Flurin Böni
Das verborgene Gold am Ende des Rainbow-Washing
Eine Analyse der Vereinbarkeit sozialen Engagements mit unternehmerischen Zielen
Chur, 2023
ISSN 1660-945X

Über die Informationswissenschaft der Fachhochschule Graubünden

Die Informationswissenschaft ist in der Schweiz noch ein relativ junger Lehr- und Forschungsbereich. International weist diese Disziplin aber vor allem im anglo-ameri-kanischen Bereich eine jahrzehntelange Tradition auf. Die klassischen Bezeichnungen dort sind Information Science, Library Science oder Information Studies. Die Grundfragestellung der Informationswissenschaft liegt in der Betrachtung der Rolle und des Umgangs mit Information in allen ihren Ausprägungen und Medien sowohl in Wirtschaft und Gesellschaft. Die Informationswissenschaft wird in Chur integriert betrachtet.

Diese Sicht umfasst nicht nur die Teildisziplinen Bibliothekswissenschaft, Archiv-wissenschaft und Dokumentationswissenschaft. Auch neue Entwicklungen im Bereich Medienwirtschaft, Informations- und Wissensmanagement und Big Data werden gezielt aufgegriffen und im Lehr- und Forschungsprogramm berücksichtigt.

Der Studiengang Informationswissenschaft wird seit 1998 als Vollzeitstudiengang in Chur angeboten und seit 2002 als Teilzeit-Studiengang in Zürich. Seit 2010 rundet der Master of Science in Business Administration das Lehrangebot ab.

Der Arbeitsbereich Informationswissenschaft vereinigt Cluster von Forschungs-, Entwicklungs- und Dienstleistungspotenzialen in unterschiedlichen Kompetenzzentren:

- Information Management & Competitive Intelligence
- Collaborative Knowledge Management
- Information and Data Management
- Records Management
- Library Consulting
- Information Laboratory
- Digital Education

Diese Kompetenzzentren werden im Swiss Institute for Information Science (SII) zusammengefasst.

Impressum

Impressum

FHGR - Fachhochschule
Graubünden
Information Science
Pulvermühlestrasse 57
CH-7000 Chur

www.informationscience.ch
www.fhgr.ch
ISSN 1660-945X

Institutsleitung

Prof. Dr. Ingo Barkow
Telefon: +41 81 286 24 61
Email: ingo.barkow@fhgr.ch

Sekretariat

Telefon: +41 81 286 24 24
Fax: +41 81 286 24 00
Email: clarita.decurtins@fhgr.ch