Hochschule für Technik
und Wirtschaft Berlin
University of Applied Sciences

Eignung von Large Language Models (LLMs) zur Generierung von Python Code zur Datenanalyse

Bachelorarbeit/Masterarbeit

Name des Studiengangs
Wirtschaftsinformatik

Fachbereich 4

vorgelegt von
Maurice Krüger

Datum:
Berlin, 10. 01. 2025

Erstgutachter: Prof. Dr.-Ing. Ingo Claßen
Zweitgutachter: Prof. Dr. Axel Hochstein

Abstract Diese Bachelorarbeit untersucht die Eignung moderner Large Language Models (LLMs) für die automatisierte Generierung von Python-Code im Kontext typischer Datenanalyseaufgaben. Dabei wird zunächst ein Überblick über die theoretischen Grundlagen von LLMs und der Programmiersprache Python gegeben. Anschließend werden in einer empirischen Untersuchung Codebeispiele durch LLMs erzeugt und mit manuell geschriebenen Skripten verglichen. Die Bewertung erfolgt anhand mehrerer Kriterien wie Korrektheit, Performanz sowie Verständlichkeit des generierten Codes. Die Ergebnisse zeigen, dass LLMs bereits in der Lage sind, einfachen bis mittleren Anforderungen in der Datenanalyse gerecht zu werden. Allerdings treten insbesondere bei komplexeren Analysen und Datenvorverarbeitungsschritten Fehlerrisiken sowie Wartbarkeitsprobleme auf. Abschließend werden Empfehlungen für den praktischen Einsatz von LLMs in der Datenanalyse abgeleitet sowie ein Ausblick auf zukünftige Entwicklungen gegeben.

Inhaltsverzeichnis

1 Einleitung

1.1 Problemstellung und Forschungsfragen

Die schnelle Entwicklung von Large Language Models (LLMs), wie zum Beispiel ChatGPT, hat in den letzten Jahren sowohl im privaten als auch im beruflichen Bereich viel Aufmerksamkeit erregt. Ursprünglich wurden LLMs hauptsächlich zur Lösung alltäglicher Probleme und der Verarbeitung und Erzeugung menschlicher Sprache eingesetzt, doch zunehmend zeigt sich, dass sie auch Programmiercode in verschiedenen Sprachen erstellen können. Besonders in der Programmiersprache Python – einer weit verbreiteten Sprache für Datenanalyse und Machine Learning – sind die Fortschritte in der automatisierten Code-Generierung durch LLMs bereits bemerkenswert[1, 2].

Aktuelle Forschungsarbeiten konzentrieren sich auf die systematische Bewertung von solch generierten Codes, um Fehlerquellen und Qualitätsmerkmale zu bemessen. Die Bereitstellung öffentlicher Evaluierungsdatensätze und -frameworks, wie etwa *HumanEval*[2] oder *EvalPlus*[3], ermöglicht standardisierte Vergleichsstudien verschiedener LLMs. Dies eröffnet neue Anwendungsfelder im Bereich der Datenanalyse: Anstatt den Code manuell zu schreiben, könnten Nutzer in Zukunft lediglich ihre Anforderungen in natürlicher Sprache formulieren und das Modell würde diese für den Nutzer umsetzen[4].

Vor diesem Hintergrund stellt sich die Frage, ob und inwiefern LLMs tatsächlich qualitativ hochwertigen Python-Code für datenanalytische Aufgaben erzeugen können und wie dieser Code im Vergleich zu manuell geschriebenen Code abschneidet. Auch die möglichen Grenzen dieser automatisierten Generierung, wie etwa in Bezug auf Performanz, Wartbarkeit oder Fehlerraten, sind hierbei von großer Bedeutung[5].

Daraus ergibt sich die zentrale **Hauptforschungsfrage**:

> *Inwieweit eignen sich Large Language Models (LLMs) zur Durchführung gängiger Datenanalyseaufgaben in Python, und wie schneidet dieser Code im Vergleich zu manuell geschriebenen Code hinsichtlich Effizienz, Korrektheit und Wartbarkeit ab?*

Zur weiteren Strukturierung dieser Hauptfrage werden mehrere Unterfragen hinzugezogen:

- **Qualität & Korrektheit:** Wie qualitativ hoch ist dieser generierte Code hinsichtlich Syntax und Implementierung von Analyseaufgaben (z. B. Datenbereinigung, Modellierung)?
- **Effizienz & Performanz:** Inwieweit entspricht der automatisch erzeugte Code modernen Standards bezüglich Laufzeit und Ressourcenverbrauch?
- **Wartbarkeit & Verständlichkeit:** Wie gut lässt sich der generierte Code verstehen, dokumentieren und erweitern?
- **Einsatzgebiete & Grenzen:** Für welche spezifischen Aufgaben in der Datenanalyse ist der Einsatz von LLMs sinnvoll und wo stoßen diese an ihre Grenzen?

1.2 Relevanz der Thematik

Die Fähigkeit, Programmiercode mit Hilfe von LLMs zu erstellen, könnte die Entwicklungsprozesse erheblich beschleunigen und neue Nutzergruppen anziehen, die bisher wenig Erfahrung mit Programmierung haben. Besonders in der Datenanalyse können viele Arbeitsschritte – vor allem wiederkehrende Aufgaben wie das Erstellen von Standard-Pipelines oder simpler Boilerplate-Code – automatisiert werden. Gleichzeitig gibt es jedoch Herausforderungen in Bezug auf Performanz, Wartbarkeit und Transparenz.

1.3 Zielsetzung

Das Ziel dieser Arbeit ist es, herauszufinden, wie gut moderne LLMs (Large Language Models) für die automatische Code-Generierung in der Datenanalyse mit Python geeignet sind. Dafür wird in einem Experiment Code von einem LLM generiert und mit manuell geschriebenem Code verglichen. Der Vergleich basiert auf Kriterien wie Korrektheit, Performance und Wartbarkeit. Auf Basis der Ergebnisse werden Empfehlungen für den Einsatz von LLMs in der Praxis gegeben und deren Grenzen diskutiert. Zum Schluss wird ein Ausblick darauf gegeben, wie sich diese Technologie in Zukunft weiterentwickeln könnte und welche Auswirkungen das auf Aufgaben in der Datenanalyse haben könnte[2, 3].

1.4 Aufbau der Arbeit

Nach dieser Einleitung (Kapitel 1) folgt in Kapitel 2 eine Darstellung der **Grundlagen**. Kapitel 3 gibt einen Überblick über den aktuellen Stand der Forschung, in dem verschiedene LLM-Modelle, Publikationen und Evaluationstechniken vorgestellt werden. Darauf aufbauend wird in Kapitel 4 die **Methodik** der Arbeit erläutert. Kapitel 5 enthält dann die **Auswertung** der gewonnenen Daten sowie den Vergleich von durch ein LLM generierten und manuell erstellten Code. Kapitel 6 fasst die Ergebnisse zusammen, beantwortet die Forschungsfragen und gibt einen **Ausblick** auf weitere Entwicklungen. Schließlich enthält Kapitel 7 den **Anhang**, einschließlich Literaturverzeichnis und relevanter Dokumentationen.

2 Grundlagen

Im folgenden Kapitel werden die theoretischen und technischen Grundlagen vorgestellt, die für das Verständnis dieser Arbeit notwendig sind. Abschnitt 2.1 beschäftigt sich mit den Large Language Models, ihrer Funktionsweise und ihrer Bedeutung in der Code-Generierung. Danach wird in Abschnitt 2.2 das Potenzial der Programmiersprache Python für die Datenanalyse erläutert, bevor Abschnitt 2.3 das Konzept der automatisierten Code-Generierung behandelt.

2.1 Einführung in Large Language Models

TODO: Aufbau aendern und umschreiben

2.1.1 Grundlegendes Konzept und aktuelle Entwicklungen

Bei großen Sprachmodellen (LLMs) handelt es sich um KI-Systeme, die mithilfe moderner Deep-Learning-Methoden entwickelt wurden, um natürliche Sprache zu verstehen und selbst Texte zu generieren [6]. Dazu gehören beispielsweise Modelle wie GPT oder spezialisierte LLMs für die Code-Generierung, die auf Transformer-Architekturen basieren und sowohl für Text- als auch Code-Anwendungen optimiert sind [1, 4, 7]. Allerdings steigt mit der zunehmenden Größe dieser Modelle, auch der Bedarf an Rechenressourcen und großen Mengen an Trainingsdaten. In den letzten Jahren wurden mehrere Benchmarks und Evaluierungsdatensätze speziell für die Code-Generierung entwickelt. Beispiele dafür sind *HumanEval*[2] und *EvalPlus*[3], die genutzt werden, um die Genauigkeit und Zuverlässigkeit von LLMs in verschiedenen Programmiersprachen zu testen. Erste Studien zeigen, dass LLMs einfache bis mittelschwere Aufgaben oft vollständig lösen können. Bei komplexeren oder sehr speziellen Aufgabenbereichen stoßen sie aber noch an ihre Grenzen[5].

2.1.2 Anwendung in der Python-Programmierung

Obwohl LLMs in vielen Sprachen Code generieren können, hat sich Python als einer der Hauptfoki herauskristallisiert. Dies liegt an der weit verbreiteten Nutzung von Python in Wissenschaft und Industrie, insbesondere in den Bereichen Datenanalyse und Machine Learning. Die umfangreichen Bibliotheken wie NumPy, pandas und scikit-learn sind Teil der Trainingskorpora, wodurch LLMs häufig in der Lage sind, Standardroutinen oder Bibliotheksfunktionen korrekt anzuwenden[2].

2.2 Einführung in Python für die Datenanalyse

2.2.1 Bedeutung und Bibliotheken

Python ist dank seiner Syntax und aktiven Community eine der am weitesten verbreiteten Sprachen für Datenanalyse [5]. Wichtige Bibliotheken wie:

- **pandas** – Datenstrukturen und -bearbeitung,
- **NumPy** – numerische Berechnungen,
- **scikit-learn** – Machine-Learning-Algorithmen,
- **Matplotlib**, **seaborn** – Visualisierung,

stellen ein reichhaltiges Ökosystem dar, das die effiziente Umsetzung datengetriebener Projekte ermöglicht. Viele davon werden bereits in LLM-Trainings berücksichtigt, wodurch generierter Code auf bekannte Funktionen zurückgreifen kann [3].

2.2.2 Typische Schritte einer Datenanalyse

Eine klassische Datenanalyse in Python kann grob in sechs Schritte unterteilt werden:

1. *Datenimport* (z. B. CSV-Dateien, Datenbanken, APIs),

2. *Datenbereinigung* (fehlende Werte, Duplikate, Datentypen),
3. *Explorative Analyse und Visualisierung* (Statistiken, Plots),
4. *Feature Engineering* (Neue Variablen, Skalierung, Kodierung),
5. *Modellierung* (Trainieren und Evaluieren von ML-Modellen),
6. *Kommunikation* (Ergebnisse präsentieren, Dokumentation).

Im Rahmen dieser Arbeit wird untersucht, ob LLMs diese Schritte automatisieren können und an welchen Stellen manuell eingegriffen werden muss [2, 3].

2.3 Automatisierte Code-Generierung für Datenanalyse

2.3.1 Funktionsweise und Vorteile

Automatisierte Code-Generierung mithilfe von LLMs basiert auf *Prompts*, also Benutzeranfragen in natürlicher Sprache. Im Gegensatz zu traditionellen Code-Generatoren, die häufig starre Templates oder regellastige Systeme verwenden, können LLMs sich flexibel an den Kontext anpassen [2]. Insbesondere in datenanalytischen Szenarien, in denen standardisierte Skripte (z. B. für das Einlesen und Bereinigen von Daten) immer wieder benötigt werden, kann dies zu einer erheblichen Zeitersparnis führen.

2.3.2 Herausforderungen und Grenzen

Trotz beeindruckender Fortschritte stößt die automatisierte Code-Generierung noch häufig an Grenzen [4, 5]:

- **Komplexe Datenstrukturen**: LLMs zeigen teils Schwächen bei Aufgaben mit hochgradiger Komplexität oder domänenspezifischem Wissen.
- **Performanz**: Generierter Code ist nicht immer optimal hinsichtlich Laufzeit oder Speicherverbrauch.
- **Wartbarkeit**: Kommentare, klare Code-Struktur und Dokumentation fehlen häufig.
- **Fehleranfälligkeit**: Auch Code, der zunächst lauffähig erscheint, kann subtile Bugs oder Sicherheitslücken enthalten.

Wie stark diese Faktoren in der Praxis ins Gewicht fallen, wird in den kommenden Kapiteln anhand einer empirischen Untersuchung (LLM-generierter vs. manuell erstellter Code) analysiert.

3 LLMs in der Programmierung – aktueller Stand

Die Entwicklung von Large Language Models (LLMs) hat in den letzten Jahren nicht nur die Art und Weise, wie natürliche Sprache verarbeitet und generiert wird, verändert, sondern auch große Fortschritte in der automatisierten Code-Erstellung ermöglicht. Durch die Kombination aus leistungsstarken Modellarchitekturen wie Transformers, großen Mengen an Trainingsdaten und moderner Hardware sind LLMs heute fester Bestandteil vieler Bereiche der Softwareentwicklung und Datenanalyse[7]. In diesem Kapitel werden die aktuellen Entwicklungen und verfügbaren Modelle vorgestellt. Außerdem wird ein Überblick über ihre Einsatzmöglichkeiten in der Softwareentwicklung und Datenanalyse gegeben. Zum Schluss werden wichtige Studien und Arbeiten zur Code-Generierung betrachtet, darunter etwa die von Chen et al. (2021) vorgestellte Arbeit zu Codex, einem Modell, das speziell für die automatisierte Programmierung entwickelt wurde [2]. TODO: umschreiben

3.1 Überblick und Vergleich von verschiedenen LLMs

Derzeit existiert eine Vielzahl an LLMs, darunter auch viele, die gezielt zur Code-Generierung entwickelt wurden. Zu den bekanntesten Beispielen zählen ChatGPT (GPTo1 als das modernste Modell), OpenAI Codex, Code Llama[8], StarCoder [9], CodeT5[5] oder CodeGen[4]. Diese Modelle teilen sich häufig folgende Merkmale:

1. **Transformer-Architektur**: Nahezu alle modernen LLMs beruhen auf dem Transformer-Modell, das mithilfe von Self-Attention Mechanismen Zusammenhänge in sequentiellen Daten (Text/Code) erfassen kann[7].

2. **Große Parameteranzahl**: Typische LLMs verfügen über hunderte Millionen bis mehrere Milliarden Parameter und benötigen entsprechend umfangreiche Trainingsdaten, zu denen in vielen Fällen öffentlich verfügbare Code-Repositories (z. B. GitHub) zählen. TODO: umschreiben

3. **Breite Sprachenunterstützung**: Neben Python werden häufig C++, Java, JavaScript und andere Programmiersprachen abgedeckt. Python nimmt jedoch oft eine zentrale Rolle ein, da sie im Bereich Datenanalyse und Machine Learning weit verbreitet ist. TODO: Umschreiben und Quelle hinzufuegen

Ein Vergleich der LLMs lässt sich anhand verschiedener Kriterien vornehmen:

- **Größe und Trainingsdaten**: Modelle wie GPT-4 oder Code Llama sind mit einer Vielzahl an Code-Datensätzen trainiert und erreichen dadurch in Benchmarks eine hohe Treffsicherheit. TODO: Quelle hinzufuegen

- **Lizenz und Offenheit**: Während GitHub Copilot und ChatGPT proprietär sind, existieren mit Code Llama, StarCoder oder CodeGen auch teils offene bzw. frei nutzbare Alternativen.

- **Spezialisierung**: Einige Modelle sind speziell auf Code-Generierung abgestimmt (z. B. Code Llama-Python, StarCoder), während andere (z. B. ChatGPT) einen generellen Sprachkontext haben, der sich jedoch auch auf Code-Aufgaben anwenden lässt.

3.2 Einsatzgebiete von LLMs in der Programmierung

Die zunehmende Leistungsfähigkeit von Large Language Models (LLMs) ermöglicht es, Programmieraufgaben in diversen Bereichen zu automatisieren oder zu beschleunigen. Häufig genannte *Einsatzgebiete* sind dabei: TODO: Quellen checken

- **Autovervollständigung und Boilerplate-Code**: Integriert in Entwicklungsumgebungen wie Visual Studio Code oder JetBrains können Tools wie GitHub Copilot Standardroutinen vorschlagen und repetitive Abläufe deutlich verkürzen [**Li2022AlphaCode**].
- **Refactoring und Fehlersuche**: Dank ihrer Kontextsensitivität können LLMs bestehenden Quellcode analysieren und an einigen Stellen optimierte oder korrigierte Varianten vorschlagen [**Zhang2023CodexRevisited**]. Dadurch lassen sich potenzielle Bugs oder ineffiziente Strukturen frühzeitig erkennen.
- **Automatisierte Dokumentation und Code-Kommentierung**: Modelle wie ChatGPT oder Code Llama-Python bieten die Möglichkeit, vorhandenen Code zu erklären oder zu kommentieren, was die Wartung und Teamkommunikation verbessert [**Phung2023MultiTask**].
- **Datenanalyse und Machine Learning**: Im Fokus vieler Python-Anwender steht die Datenverarbeitung mit Bibliotheken wie `pandas`, `NumPy` oder `scikit-learn`. LLMs können hier einfache Skripte für das Einlesen, Transformieren und Visualisieren von Daten generieren [**Wang2023DataCentric**], wodurch gerade Einsteiger schnell produktiv werden.
- **Rapid Prototyping**: In frühen Entwicklungsphasen nutzen Entwickler LLMs als Assistent, um zügig Prototypen zu erstellen. Die Modelle liefern Vorschläge für Projektstrukturen, Abhängigkeiten und Testumgebungen.

Obwohl diese Einsatzgebiete großes Potenzial bieten, sind LLMs nicht fehlerfrei. Gerade bei komplexen architektonischen Entscheidungen oder domänenspezifischen Anforderungen ist das *menschliche Fachwissen* weiterhin essenziell, um die Qualität und Wartbarkeit des Codes zu gewährleisten.

3.3 Vergangene Studien und Arbeiten zur Code-Generierung

Die Forschung zur automatisierten Code-Generierung hat in den letzten Jahren eine rasante Entwicklung erlebt, wobei Arbeiten aus den Bereichen *Software Engineering*, *Natural Language Processing* und *Machine Learning* zusammenfließen. Im Folgenden werden einige zentrale Punkte hervorgehoben:

1. **Frühe Ansätze und Expertensysteme** Bereits in den 1980er-Jahren experimentierten Wissenschaftler mit regelbasierten Generatoren, die für bestimmte Domänen (z. B. Datenbankzugriffe) begrenzte Codefragmente erzeugten. Diese Systeme waren allerdings sehr spezialisiert und kaum flexibel [**Henderson2023LLMRefactor**].

2. **Tiefe neuronale Modelle und Sequenz-zu-Sequenz-Architekturen** Mit dem Aufkommen tief lernender Methoden (LSTMs, GRUs) in der NLP-Forschung rückte die Idee in den Fokus, *natürlichsprachliche Beschreibungen* in Programmcode zu übersetzen. Spätere Transformer-basierte Modelle wie GPT, Codex oder Code Llama zeigten, dass eine größere Datenbasis (z. B. GitHub-Repositories) die Qualität der Code-Generierung erheblich steigern kann [**AlphaCodeDeepMind**].

3. **Benchmarks und Evaluierungen** Um generierten Code objektiv zu messen, wurden standardisierte Benchmarks wie *HumanEval* [**OpenAI2021**] oder *EvalPlus* [**EvalPlus2023**] entwickelt. Sie testen, inwieweit LLMs funktional korrekten und performant lauffähigen Code liefern. Aktuelle Studien erweitern diese Benchmarks auf komplexere Szenarien, z. B. Wettbewerbsniveau („AlphaCode“) [**Li2022AlphaCode**] oder domänenspezifische Datenanalyse [**Wang2023DataCentric**].

4. **Aktuelle Forschungsschwerpunkte** Zeitgenössische Arbeiten untersuchen u. a. die *Fehlerquote* (z. B. Syntaxfehler vs. semantische Fehler), die *Wartbarkeit* (z. B. Code-Kommentierung) sowie die *Performanz* (z. B. Laufzeit, Speicherverbrauch) des generierten Codes. Insbesondere in größeren Projekten bleibt menschliches Eingreifen unverzichtbar, da LLMs gerade bei sehr umfangreichen oder domänenspezifischen Anwendungen an ihre Grenzen stoßen [**Zhang2023CodexRevisited**].

Zusammenfassend zeigen die bisherigen Studien, dass LLMs zwar weitreichendes Potenzial zur *automatisierten Code-Generierung* besitzen, jedoch kein vollständiger Ersatz für erfahrene Entwickler sind. Insbesondere bei sicherheitskritischen Anwendungen oder stark branchenspezifischen Projekten empfiehlt sich ein sorgfältiges Code-Review durch Experten. Nichtsdestotrotz deutet die Entwicklung darauf hin, dass LLMs das Potenzial haben, den Programmieralltag nachhaltig zu verändern – vom automatisierten Erzeugen kleiner Module bis hin zur Unterstützung bei komplexen Datenanalyseprozessen.

4 Methodik

4.1 Vorgehensweise der Untersuchung

In der Untersuchung soll geprüft werden, inwieweit Large Language Models in der Lage sind gängige Datenanalyse-Schritte auf Grundlage eines gegebenen Datensatzes durchzuführen. Hierbeui wird ChatGPT als aktueller Marktführer mit dem Sprachmodell GPTo1 verwendet, welches das neueste Modell ist. Ebenso gilt es herauszufinden wie qualitativ und effizient diese Lösung ist. Hierbei bezieht es sich auf die Forschungsfragen aus Kapitel 1 1.1. Die Vorgehensweise hierbei ist wie folgendermaßen: Zuerst wird der verwendete Datensatz von Berlin Open Data an das Modell übergeben und dazu eine Prompt. Diese Prompts können in Kapitel 4.2 4.2 eingesehen werden. Anschließend wird der generierte Code mit HumanEval evaluiert um die generelle Qualität des Codes zu bewerten. Daraufhin wird der Code ausgeführt und die benötigte Laufzeit gemessen. Abschließend werden die Ergebnisse des Ausführung bewertet. (TODO: Tool zur Auswertung der Ergebnisse oder Vergleich mit manuellem Skript) Die Ergebnisse der Auswertung mit *EvalPlus*[3] werden in Kapitel 5 **??** detailliert dargestellt und

auch mit den Ergebnissen anderer Arbeiten, wie etwa von Chen et al. (2021)[2] und Liu et al. (2023)[1] verglichen.

4.2 Testfälle der Datenanalyse

4.2.1 Testfall 1

Im ersten Testfall soll der Datensatz nach einer gewissen Spalte sortiert werden. Die Begründung hierfür ist, dass dies eine sehr einfache, aber auch sehr häufig aufkommende Datenanalyse-Aufgabe ist und somit einen guten Einstieg in die Untersuchung darstellt. Die Prompt für diese Aufgabe lautet: *Erstelle mir ein Python Skript, mit welchem der Datensatz nach der Anzahl der Straftaten insgesamt eines Bezirks sortiert wird..*

4.2.2 Testfall 2

Aufbauend auf Testfall eins, soll im zweiten Testfall eine simple Visualisierung des Datensatzes stattfinden. Es soll ein Balkendiagramm der Bezirke und wieder der Straftaten insgesamt erstellt werden. Hierbei sollen die Bezirke auf der x-Achse und die Anzahl der Straftaten auf der y-Achse dargestellt und absteigend mit der Anzahl der Bezirke sortiert werden. Die Prompt für diese Aufgabe lautet: *Erstelle mir ein Python Skript, mit welchem ein Balkendiagramm der Anzahl der Straftaten insgesamt pro Bezirk erstellt wird. Hierbei sollen auf der X-Achse die Bezirke und auf der Y-Achse die Anzahl der Straftaten sein..*

4.2.3 Testfall 3

Im dritten Testfall soll das Sprachmodell die prozentuale Verteilung der Straftaten in den Bezirken berechnen. Die Prompt für diese Aufgabe lautet: *Erstelle mir ein Python Skript, mit welchem die prozentuale Verteilung der genauen Straftaten anteilig der Straftaten insgesamt pro Bezirk berechnet wird..*

4.3 Auswertungskriterien

Die Auswertung der generiertes Python Skripte erfolgt anhand der in Kapitel 1 1.1 definierten Kriterien. Hierfür wird zum einen das Tool *EvalPlus*[3] verwendet, welches eine standardisierte Bewertung zur Korrektheit des generierten Codes ermöglicht. Ebenso werden Laufzeit und Ressourcennutzung des Codes bewertet, um die Effizienz des Skripts zu bemessen. Anschließend wird der Code manuell auf Wartbarkeit und Verständlichkeit geprüft. Hierbei wird insbesondere auf die Strukturierung des Codes, die Kommentierung und die Dokumentation geachtet. Die Ergebnisse der Auswertung werden in Kapitel 5 **??** detailliert dargestellt und bewertet.

4.4 Verwendete Tools

1. **Large Language Model**: Als Large Language Model wird ChatGPT mit GPTo1 verwendet, da ChatGPT als aktueller Marktführer gilt und GPTo1 das neueste Modell ist.

2. **Evaluierungsframework**: Zur Bewertung der generierten Python Skripte wird das Evaluierungsframework *EvalPlus*[3] verwendet, welches eine standardisierte Bewertung zur Korrektheit des generierten Codes ermöglicht.

Literatur

[1] Jiawei Liu u. a. “Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation”. In: *Advances in Neural Information Processing Systems*. Hrsg. von A. Oh u. a. Bd. 36. Curran Associates, Inc., 2023, S. 21558–21572. URL: https://proceedings.neurips.cc/paper_files/paper/2023/file/43e9d647ccd3e4b7b5baab53f0368686-Paper-Conference.pdf.

[2] Mark Chen u. a. *Evaluating Large Language Models Trained on Code*. 2021. arXiv: 2107.03374 [cs.LG]. URL: https://arxiv.org/abs/2107.03374.

[3] Jiawei Liu u. a. “Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation”. In: *Thirty-seventh Conference on Neural Information Processing Systems*. 2023. URL: https://openreview.net/forum?id=1qvx610Cu7.

[4] Erik Nijkamp u. a. *CodeGen: An Open Large Language Model for Code with Multi-Turn Program Synthesis*. 2023. arXiv: 2203.13474 [cs.LG]. URL: https://arxiv.org/abs/2203.13474.

[5] Yue Wang u. a. *CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation*. 2021. arXiv: 2109.00859 [cs.CL]. URL: https://arxiv.org/abs/2109.00859.

[6] Humza Naveed u. a. *A Comprehensive Overview of Large Language Models*. 2024. arXiv: 2307.06435 [cs.CL]. URL: https://arxiv.org/abs/2307.06435.

[7] Cem Subakan u. a. “Attention Is All You Need In Speech Separation”. In: *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2021, S. 21–25. DOI: 10.1109/ICASSP39728.2021.9413901.

[8] Baptiste Rozière u. a. *Code Llama: Open Foundation Models for Code*. 2024. arXiv: 2308.12950 [cs.CL]. URL: https://arxiv.org/abs/2308.12950.

[9] Raymond Li u. a. *StarCoder: may the source be with you!* 2023. arXiv: 2305.06161 [cs.CL]. URL: https://arxiv.org/abs/2305.06161.