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, 05. 01. 2025

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

Inhaltsverzeichnis

1 Einleitung

1.1 Problemstellung und Forschungsfragen

Die rasante Entwicklung von Large Language Models (LLMs), wie beispielsweise ChatGPT, hat in den vergangenen Jahren sowohl im privaten als auch im beruflichen Umfeld für große Aufmerksamkeit gesorgt. Während LLMs ursprünglich vor allem zur Verarbeitung und Generierung natürlicher Sprache eingesetzt wurden, zeigt sich zunehmend, dass sie auch Programmiercode in diversen Sprachen erzeugen können. Insbesondere für Python – eine häufig genutzte Sprache für Datenanalyse und Machine Learning – sind die Fortschritte in der automatisierten Code-Generierung bereits beachtlich [1, 2].

Aktuelle Forschungsarbeiten befassen sich mit der systematischen Evaluation solcher Code-Generierungen, um Fehlerquellen und Qualitätsmerkmale quantifizieren zu können [3, 4]. Die Bereitstellung öffentlicher Evaluierungsdatensätze und -frameworks, wie zum Beispiel *HumanEval* oder *EvalPlus*, ermöglicht standardisierte Vergleichsstudien verschiedener LLMs. Dies eröffnet neue Anwendungsfelder im Bereich der Datenanalyse: Anstatt den Code manuell zu schreiben, könnten Anwender in Zukunft lediglich ihre Anforderungen in natürlicher Sprache formulieren und vom Modell automatisch umsetzen lassen [5].

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 erstellten Skripten abschneidet. Auch mögliche Grenzen dieser automatisierten Generierung, etwa in Bezug auf Performanz, Wartbarkeit oder Fehlerraten, spielen hierbei eine zentrale Rolle [6].

Daraus ergibt sich die zentrale **Hauptforschungsfrage**:

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

Zur weiteren Strukturierung dieser Hauptfrage werden mehrere Unterfragen hinzugezogen:

- **Qualität & Korrektheit:** Wie hoch ist die Korrektheit des generierten Codes 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 stößt die Technologie an ihre Grenzen?

1.2 Relevanz der Thematik

Die Möglichkeit, Programmiercode mithilfe von LLMs zu generieren, könnte Entwicklungsprozesse signifikant beschleunigen und neue Nutzergruppen ansprechen, die bislang nur wenig Erfahrung mit Programmierung haben. Gerade in der Datenanalyse können viele Arbeitsschritte – insbesondere repetitive Abläufe wie das Schreiben von Standard-Pipelines oder Boilerplate-Code – automatisiert werden. Gleichzeitig ergeben sich jedoch Herausforderungen bezüglich *Performanz*, *Wartbarkeit* und *Transparenz* [2].

1.3 Zielsetzung

Ziel dieser Arbeit ist eine **systematische Untersuchung**, wie gut sich moderne LLMs für die automatisierte Code-Generierung im Bereich der Python-Datenanalyse eignen. Dazu wird in einem empirischen Experiment Code durch ein LLM erzeugt und mit manuell geschriebenem Code verglichen. Dieser Vergleich erfolgt anhand definierter Kriterien wie *Korrektheit*, *Performance* und *Wartbarkeit*. Auf Basis der Ergebnisse werden Handlungsempfehlungen für den praktischen Einsatz abgeleitet und Grenzen der Technologie aufgezeigt. Ein abschließender *Zukunftsausblick* beleuchtet mögliche Weiterentwicklungen im Bereich der LLMs und deren Einfluss auf datenanalytische Aufgaben [3, 4].

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 LLM-generiertem und manuell erstelltem 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 dargelegt, die zum Verständnis dieser Arbeit erforderlich sind. Abschnitt 2.1 widmet sich den Large Language Models, deren Funktionsweise und ihrer Rolle in der Code-Generierung. Anschließend wird in Abschnitt 2.2 das Potenzial der Programmiersprache Python für Datenanalyse erläutert, bevor Abschnitt 2.3 sich dem Konzept der automatisierten Code-Generierung zuwendet.

2.1 Einführung in Large Language Models

2.1.1 Grundlegendes Konzept und aktuelle Entwicklungen

Large Language Models (LLMs) sind KI-Modelle, die mithilfe moderner Deep-Learning-Architekturen darauf trainiert werden, Sprache zu verstehen und zu generieren [2](TODO: quelle 1 auch?). Moderne Modelle wie GPT oder Code-spezifische LLMs beruhen oft auf

Transformer-Architekturen, die sich sowohl zur Text- als auch zur Code-Generierung eignen [5]. Mit zunehmender Größe der Modelle, die teils mehrere hundert Milliarden Parameter umfassen, steigt jedoch auch der Bedarf an Rechenleistung und Trainingsdaten.

Verschiedene Forschungsarbeiten haben in den letzten Jahren spezielle *Benchmarks* und *Evaluierungsdatensätze* für Code-Generierung entwickelt. Beispiele sind *HumanEval* [3] und *EvalPlus* [4], anhand derer die Genauigkeit und Robustheit der LLMs in unterschiedlichen Programmiersprachen gemessen werden kann. Erste Studien zeigen, dass LLMs bereits fähig sind, einfache bis mittelschwere Aufgaben vollständig zu lösen, während bei komplexeren und domänenspezifischen Szenarien noch deutliche Leistungsdefizite zu beobachten sind [6].

2.1.2 Anwendung in der Python-Programmierung

Obgleich LLMs in zahlreichen Sprachen Code generieren können, hat sich Python als einer der Schwerpunkte herauskristallisiert. Dies liegt an der Verbreitung von Python in Wissenschaft und Industrie, insbesondere für Datenanalyse und Machine Learning [5]. Die umfangreichen Bibliotheken (z. B. NumPy, pandas, scikit-learn) fließen in die Trainingskorpora ein, sodass LLMs häufig bereits Standardroutinen oder Bibliotheksfunktionen korrekt anwenden [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 [6]. 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 [4].

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 [3, 4].

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 [5, 6]:

- **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.

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] OpenAI. *HumanEval*. https://github.com/openai/human-eval. 2021.

[4] EvalPlus. *EvalPlus*. https://github.com/evalplus/evalplus. 2021.

[5] 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.

[6] 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.