

Überwindung der Programmierkluft in der Produktion und Fertigung

Demokratisierung der Programmierfähigkeiten von Ingenieurinnen und Ingenieuren durch GPT-Coding-Assistenten

Patrick Flore*,
Marco Hussong und
Peter M. Simon

Die Digitalisierung und KI verändern die Industrie – doch viele Ingenieurinnen und Ingenieure stehen vor einer Hürde: fehlende Programmierkenntnisse. Wie können GPT-basierte Assistenten wie GitHub Copilot dabei helfen? Dieser Beitrag beleuchtet ihre Stärken, zeigt Unterschiede zu No-Code- und Low-Code-Lösungen auf und diskutiert Herausforderungen bei der Integration in den Arbeitsalltag. Zudem wird die Bedeutung algorithmischer Problemlösungskompetenzen für die technische Praxis herausgestellt.

Motivation

Die fortschreitende Digitalisierung von Fertigungs- und Produktionssystemen hat zu einer grundlegenden Veränderung der Industrielandschaft geführt [1]. Moderne Technologien wie das Internet der Dinge (IoT), künstliche Intelligenz (KI) und die Datenanalyse werden in zunehmendem Maße in Produktionsprozesse integriert, wodurch der Bedarf an Arbeitskräften, welche sich in diesen Bereichen auskennen, steigt [2]. Ingenieure müssen sich daher zwingend mit Programmierkenntnissen auseinanderset-

zen, um digitale Systeme entwickeln, implementieren und warten zu können. Allerdings besteht eine signifikante Diskrepanz zwischen den erforderlichen Programmierkenntnissen und den aktuellen Fähigkeiten vieler Ingenieure im Fertigungssektor.

Die identifizierte Qualifikationslücke lässt sich auf mehrere Faktoren zurückführen. In den traditionellen Lehrplänen für Ingenieure wurde der Programmierung nur eine untergeordnete Rolle beigemessen, wobei der Fokus auf fachspezifischem Wissen und praktischen Anwendungen lag [3]. Erst in jüngster Zeit haben Bil-

dungseinrichtungen damit begonnen, umfassende Programmierkurse in ingenieurwissenschaftliche Studiengänge zu integrieren. Darüber hinaus werden Programmierfähigkeiten in Unternehmen häufig von spezialisierten IT-Abteilungen ausgeführt, wodurch sie von der Produktionsplanung, der Qualitätssicherung und anderen Kernteams getrennt werden. Diese Trennung erschwert die gemeinsame Entwicklung integrierter Lösungen und verlangsamt die Einführung digitaler Technologien in der Produktion.

Um diese Lücke zu schließen, haben sich No-Code- und Low-Code-Plattformen wie KNIME, Tableau und Power BI durchgesetzt. Diese Werkzeuge ermöglichen es den Benutzern, Datenanalysen durchzuführen und Anwendungen mit minimalem Programmieraufwand zu erstellen, was die Einstiegshürde für Nicht-Programmierer senkt. Während diese Plattformen für bestimmte Anwendungen von Vorteil sind, stoßen sie bei komplexeren Aufgaben wie der Integration von KI über die grundlegende Datenanalyse hinaus, der Anbindung an proprietäre Systeme

* Korrespondenzautor

Jun. Prof. Dr.-Ing. Patrick Flore; Lehrstuhl für Fertigungstechnik und Betriebsorganisation (FBK), RPTU Kaiserslautern-Landau; Postfach 3049, 67663 Kaiserslautern; Tel.: +49 (0) 631 205-2618, E-Mail: fbk@mv.rptu.de

Weitere Autoren

Marco Hussong, M. Sc.; FBK an der RPTU Kaiserslautern-Landau
Peter M. Simon, M. Sc.; FBK an der RPTU Kaiserslautern-Landau

Hinweis

Bei diesem Beitrag handelt es sich um einen von den Advisory Board-Mitgliedern des ZWF-Sonderheftes wissenschaftlich begutachteten Fachaufsatz (Peer-Review).

oder der Anpassung fortgeschrittener Funktionen an ihre Grenzen [4]. Die ausschließliche Verwendung von No-Code- und Low-Code-Lösungen kann die Entwicklung grundlegender Programmierfähigkeiten bei Ingenieuren unbeabsichtigt verzögern [5].

Das Aufkommen von KI-basierten Coding-Assistenten wie GitHub Copilot bietet eine neue Möglichkeit, Programmierkenntnisse im Fertigungssektor zu verbreiten. Diese auf großen Sprachmodellen (Large Language Models, LLMs) basierenden Assistenten sind in der Lage, auf der Grundlage von Eingabeaufforderungen in natürlicher Sprache Codeausschnitte zu generieren, kontextbezogene Fehleranalysen durchzuführen, Code zwischen Programmiersprachen zu übersetzen und sogar Dokumentationen zu erstellen. Sie fungieren als interaktive Partner, die den Benutzer bei der Problemlösung unterstützen und Programmierkonzepte in Echtzeit erlernen [6].

Diese Fähigkeiten haben das Potenzial, die Herangehensweise von Ingenieuren an Programmieraufgaben zu verändern [7]. Durch die Verkürzung der Lernkurve im Vergleich zu herkömmlichen Programmiersprachen können KI-basierte Coding-Assistenten Ingenieure in die Lage versetzen, maßgeschneiderte Lösungen zu entwickeln, die Automatisierung zu verbessern und effektiver zu Initiativen für den digitalen Wandel in ihren Unternehmen beizutragen. Darüber hinaus erleichtern sie das lebenslange Lernen und die Weiterentwicklung von Fähigkeiten, sodass Ingenieure mit dem raschen technologischen Fortschritt Schritt halten können.

Die Integration von GPT-basierten Programmierwerkzeugen in die Ingenieurpraxis ist jedoch nicht ohne Herausforderungen [8]. Komplexe Software-Architekturen und die konsequente Anwendung von Programmierparadigmen wie der objekt-orientierten Programmierung (OOP) sind nach wie vor Bereiche, in denen diese Werkzeuge Schwierigkeiten haben könnten. Es besteht auch die Gefahr, dass fehlerhafter oder suboptimaler Code erzeugt wird, insbesondere bei komplexen algorithmischen Problemen, was die Bedeutung der Förderung solider Grundkenntnisse und eines kritischen Ansatzes bei der Validierung und beim Testen von Code unterstreicht.

Funktionsweise von KI-basierten Coding-Assistenten

KI-basierte Coding-Assistenten stellen eine Art Werkzeug dar, welches den Entwicklungsprozess vereinfachen und beschleunigen soll. Die Unterstützung erfolgt durch die effiziente Erstellung von Code auf Basis unterschiedlicher Eingaben. In Abhängigkeit von der Art der Eingabe lassen sich die folgenden Unterstützungsmechanismen differenzieren. Bild 1 demonstriert exemplarische Eingaben der diversen Eingabearten anhand eines typischen ingenieurwissenschaftlichen Anwendungsfalls. Die Programmieraufgabe besteht darin, eine CSV-Datei einzulesen, eine Spalte zu selektieren und eine Fast-Fourier-Transformation der Messwerte dieser Spalte zu berechnen. Die Aufgaben können hierbei in fünf Kategorien unterteilt werden:

■ Code aus Prompts

Der Assistent generiert auf Basis einer in natürlicher Sprache formulierten Idee oder Aufgabenstellung funktionierenden Code. Dabei ist eine detaillierte Beschreibung der Problemstellung und Anforderungen erforderlich.

■ Code aus Code-Kontext und -Kommentaren

Unter der Voraussetzung, dass bereits Teile des Codes verfasst und Anforderungen in Form von Kommentaren hinterlegt wurden, ist der Assistent in der Lage, den fehlenden Code zu ergänzen. Durch die Einbeziehung des bereits angefertigten Codes erfolgt die Generierung des neuen Codes.

■ Code-Erklärung

Sofern ein Abschnitt eines Codes als unverständlich erachtet wird, besteht die Möglichkeit, den Assistenten zu Rate zu ziehen, welcher eine Klarstellung vornimmt. Diesbezüglich wird Unterstützung bei der Ergründung der Logik, Struktur und Funktionsweise des Codes geboten.

■ Fehleranalyse

Bei einer fehlerhaften Ausführung des Codes oder einer Abweichung von den erwarteten Funktionsweisen erfolgt durch den Assistenten eine Identifikation der Ursachen. Basierend auf einer Analyse des Codes werden Logikfehler, Syntaxprobleme oder andere Unstimmigkeiten aufgezeigt, woraufhin

Empfehlungen zur Behebung des Problems gegeben werden.

■ Code-Übersetzung

Der Assistent ist imstande, unterschiedliche Arten von Code-Übersetzungen vorzunehmen. Einerseits ist er imstande, Code von einer Programmiersprache in eine andere zu übertragen, beispielsweise von Python nach Java. Des Weiteren ist eine Übertragung von Code in der gleichen Programmiersprache zu unterschiedlichen Lösungskonzepten möglich. Ein Beispiel hierfür bilden die Frameworks TensorFlow und PyTorch, die für die Erstellung von Modellen des Deep Learning verwendet werden können. Eine Übersetzung der unterschiedlichen Syntax dieser beiden Konzepte kann durch KI-Coding-Assistenten unterstützt werden. Dabei ist jedoch sicherzustellen, dass die Funktionalität des ursprünglichen Codes erhalten bleibt und den definierten Anforderungen entspricht.

Unterschiede zu No-Code-/Low-Code-Systemen

Die zunehmende Digitalisierung in der Fertigungsindustrie hat sowohl zur Entwicklung von No-Code-/Low-Code-Plattformen (NC-/LC-Plattformen) als auch von KI-basierten Coding-Assistenten geführt. Beide Ansätze zielen darauf ab, die Softwareentwicklung zu vereinfachen und den Zugang für Personen mit begrenzten Programmierkenntnissen zu erleichtern. Dennoch unterscheiden sie sich in Bezug auf ihren Anwendungsbereich, ihre Flexibilität und ihre Einschränkungen.

Problemlösungsraum von NC/LC und Coding-Assistenten

No-Code-/Low-Code-Plattformen (NC-/LC-Plattformen) ermöglichen es Benutzern, über visuelle Schnittstellen Anwendungen zu erstellen, ohne tiefgehende Programmierkenntnisse. Sie eignen sich für die schnelle Entwicklung von Standardanwendungen durch vorgefertigte Module und Drag-and-Drop-Funktionen. KI-basierte Coding-Assistenten wie GPT-Modelle unterstützen Entwickler direkt beim Schreiben von Code in traditionellen Programmiersprachen. Sie generieren Code aus natürlichen Spracheingaben, vervoll-

Code aus Prompt

Schreibe Pythoncode, der eine CSV namens "experiment_01.csv" öffnet, einliest und die Fast Fourier Transformation der Spalte "X1_ActualAcceleration" berechnet, die mit einer Samplingrate von 100 ms aufgezeichnet wurde.

Um die Fast Fourier Transformation (FFT) der Spalte "X1_ActualAcceleration" aus einer CSV-Datei namens "experiment_01.csv" zu berechnen, können wir die Python-Bibliothek pandas zum Einlesen der Datei und numpy und scipy.fft für die Berechnung der FFT verwenden.

Hier ist der Code, der diese Aufgabe erfüllt:

```
python
import pandas as pd
import numpy as np
from scipy.fft import fft, fftfreq
import matplotlib.pyplot as plt

# CSV-Datei einlesen
df = pd.read_csv("experiment_01.csv")

# Samplingrate und Sampling-Intervall definieren (100 ms = 0.1 s)
sampling_interval = 0.1 # in Sekunden
sampling_rate = 1 / sampling_interval

# Beschleunigungsdaten aus der Spalte "X1_ActualAcceleration" extrahieren
acceleration_data = df["X1_ActualAcceleration"].values

# Fast Fourier Transformation berechnen
n = len(acceleration_data)
fft_result = np.fft.fft(acceleration_data)
```

Code aus Kontext und -kommentaren

Ergänze den folgenden Code:
import pandas as pd
import matplotlib.pyplot as plt

```
df = pd.read_csv("experiment_01.csv")
sampling_interval = 0.1 # in Sekunden
sampling_rate = 1 / sampling_interval
acceleration_data = df["X1_ActualAcceleration"].values

# Fast Fourier Transformation (FFT) durchführen
# Frequenzen berechnen
# Nur positive Frequenzen und Amplituden verwenden
# Ergebnis plotten
```

Um den Code zu vervollständigen, der eine Fast Fourier Transformation (FFT) durchführt, die Frequenzen berechnet, nur die positiven Frequenzen verwendet und das Ergebnis plottet, kannst du den folgenden Code verwenden:

```
python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Daten einlesen
df = pd.read_csv("experiment_01.csv")
sampling_interval = 0.1 # in Sekunden
sampling_rate = 1 / sampling_interval
acceleration_data = df["X1_ActualAcceleration"].values
```

Codeerklärung

Erkläre und kommentiere:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

df = pd.read_csv("experiment_01.csv")
sampling_interval = 0.1 # in Sekunden
sampling_rate = 1 / sampling_interval
acceleration_data = df["X1_ActualAcceleration"].values

fft_result = np.fft.fft(acceleration_data)
fft_amplitude = np.abs(fft_result)

n = len(acceleration_data)
frequencies = np.fft.fftfreq(n, d=sampling_interval)

positive_frequencies = frequencies[n//2:]
positive_amplitudes = fft_amplitude[n//2:]
```

In diesem Code-Snippet wird die Fast Fourier Transform (FFT) angewendet, um die Frequenzkomponenten von Beschleunigungsdaten zu analysieren. Lassen Sie uns die einzelnen Teile des Codes detailliert erklären und kommentieren.

Code-Analyse und Erklärung

```
python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Fehleranalyse

Der Code liefert den Fehler: `SyntaxError: unterminated string literal (detected at line 8)` (detected at line 8)
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

df = pd.read_csv("experiment_01.csv")
sampling_interval = 0.1 # in Sekunden
sampling_rate = 1 / sampling_interval
acceleration_data = df["X1_ActualAcceleration"].values

fft_result = np.fft.fft(acceleration_data)
fft_amplitude = np.abs(fft_result)

n = len(acceleration_data)
frequencies = np.fft.fftfreq(n, d=sampling_interval)

positive_frequencies = frequencies[n//2:]
positive_amplitudes = fft_amplitude[n//2:]

Der Fehler `SyntaxError: unterminated string literal (detected at line 8)` tritt auf, weil der String in der Zeile, in der die Daten geladen werden, nicht korrekt beendet wurde. In deinem Code ist der String für die Spalte "X1_ActualAcceleration" nicht richtig abgeschlossen.

Hier ist der korrigierte Code:

```
python
import pandas as pd
import numpy as np
```

Codeübersetzung

Übersetze den Pythoncode in Matlab:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

df = pd.read_csv("experiment_01.csv")
sampling_interval = 0.1 # in Sekunden
sampling_rate = 1 / sampling_interval
acceleration_data = df["X1_ActualAcceleration"].values

fft_result = np.fft.fft(acceleration_data)
fft_amplitude = np.abs(fft_result)

n = len(acceleration_data)
frequencies = np.fft.fftfreq(n, d=sampling_interval)

positive_frequencies = frequencies[n//2:]
positive_amplitudes = fft_amplitude[n//2:]
```

Hier ist die Übersetzung des gegebenen Python-Codes in MATLAB:

```
matlab
% Laden der Daten
df = readtable('experiment_01.csv');
sampling_interval = 0.1; % in Sekunden
sampling_rate = 1 / sampling_interval;

% Beschleunigungsdaten extrahieren
acceleration_data = df.X1_ActualAcceleration;
```

Bild 1. Übersicht über die Funktionsweise von KI-basierten Coding-Assistenten

ständigen Code, analysieren Fehler und übersetzen Code zwischen Sprachen oder Frameworks. Dadurch ermöglichen sie die Erstellung komplexer, maßgeschneiderter Lösungen, die über die Möglichkeiten von NC/LC-Plattformen hinausgehen.

Limitierungen von NC/LC

Obwohl NC/LC-Plattformen den Entwicklungsprozess vereinfachen, weisen sie mehrere Einschränkungen auf.

Begrenzte Flexibilität und eingeschränkte Kontrolle über den Quellcode

NC/LC-Plattformen beschleunigen den Entwicklungsprozess und sind zugänglich für Benutzer ohne Programmiererfahrung. Allerdings sind sie in ihrer Flexibilität eingeschränkt, da Entwickler kein vollen Zugriff auf den zugrunde liegenden Quellcode und die Metamodelle haben. Dies erschwert detaillierte Anpassungen und die Entwicklung komplexer Softwareproduktli-

nien [9]. Zudem fehlen häufig ausreichende Test- und Qualitätssicherungswerkzeuge, was die Bewältigung komplexer Anforderungen weiter erschwert [10]. In Unternehmensumgebungen, die hochgradig angepasste Lösungen erfordern, stoßen NC/LC-Plattformen daher an ihre Grenzen.

Vendor Lock-in

Ein zentrales Problem von NC/LC-Plattformen ist der Vendor Lock-in, der die Migration von Anwendungen auf andere Systeme erschwert und langfristige Kosten verursachen kann. Strategien zur Bewältigung dieses Problems umfassen die Nutzung von Open-Source-NC/LC-Plattformen wie BESSER, um Abhängigkeiten von Anbietern zu reduzieren [11], sowie den Einsatz von Middleware-Lösungen zur Abstraktion von Diensten und Erleichterung der Datenmigration zwischen verschiedenen Cloud-Anbietern [12].

Leistungs- und Skalierungsprobleme

Aufgrund ihrer Abstraktionsschichten sind NC/LC-Plattformen oft nicht für leistungsintensive oder hochskalierbare Anwendungen geeignet. Die eingeschränkte Kontrolle über den Quellcode behindert Optimierungen, was insbesondere in Unternehmensumgebungen mit hohen Anforderungen an Zuverlässigkeit und Effizienz problematisch ist [9, 10].

Limitierungen von KI-Coding-Assistenten

Auch KI-basierte Coding-Assistenten haben ihre Grenzen. Diese lassen sich in vier Hauptprobleme unterteilen.

Qualität und Zuverlässigkeit des Codes

KI-generierter Code kann Fehler enthalten, ineffizient sein oder gegen Best Practices verstoßen, weshalb eine sorgfältige Überprüfung durch erfahrene Entwickler erforderlich ist. Studien zeigen, dass

Tools wie *GitHub Copilot* und *ChatGPT* zwar oft korrekten Code generieren, die Qualität jedoch variiert und menschliche Aufsicht notwendig bleibt [13, 14].

Kontextverständnis

KI-basierte Coding-Assistenten verstehen oft nicht den spezifischen Kontext eines Projekts, was zu irrelevanten oder suboptimalen Vorschlägen führen kann. Dies liegt daran, dass sie auf verallgemeinerten Datensätzen trainiert sind und nuancierte Projektanforderungen nicht immer erfassen. Studien zeigen eine abnehmende Effektivität bei komplexen Abhängigkeiten und Schwierigkeiten in Umgebungen wie Jupyter-Notebooks [15, 16, 18].

Abhängigkeit von Trainingsdaten

Die Leistung von KI-basierten Coding-Assistenten hängt stark von ihren Trainingsdaten ab. Veraltete oder unsichere Programmierpraktiken in diesen Daten können zu fehlerhaftem oder anfälligem Code führen. Zudem besteht das Risiko von Angriffen durch böswillige Trainingsbeispiele [17, 18].

Erfordernis von grundlegenden Programmierkenntnissen:

Trotz ihrer Unterstützung ersetzen KI-basierte Coding-Assistenten nicht die Notwendigkeit grundlegender Programmierkenntnisse. Fähigkeiten wie Problemlösung und Debugging sind weiterhin essenziell, um KI-Tools effektiv zu nutzen. Es besteht die Gefahr, dass sich Anfänger zu sehr auf diese Tools verlassen und dabei wichtige Grundkonzepte nicht vollständig verstehen.[19, 20]

Zusammenfassung und Ausblick

Die digitale Transformation der Fertigungsindustrie bedingt eine verstärkte Einbindung von Programmierkenntnissen in ingenieurwissenschaftliche Studiengänge, um den steigenden Anforderungen an die Berufsgruppe gerecht zu werden. Die traditionelle Trennung zwischen IT-Abteilungen und Kernteams sowie die begrenzte Programmierausbildung in ingenieurwissenschaftlichen Studiengängen haben zu einer signifikanten Diskrepanz zwischen den erforderlichen und den tatsächlich vorhandenen Qualifikationen geführt. Obgleich No-Code-/Low-Code-Platt-

formen einen ersten Schritt zur Überbrückung der identifizierten Defizite darstellen, zeigen sie sich bei komplexeren Anforderungen als unzureichend und können die Entwicklung tiefergehender Programmierkompetenzen verzögern.

In diesem Kontext erweisen sich GPT-basierte Coding-Assistenten als vielversprechende Alternative. Ihre Kompetenz, Code aus natürlichen Spracheingaben zu generieren, Fehler zu analysieren und Code zwischen verschiedenen Sprachen zu übersetzen, erlaubt es Ingenieuren, sich unmittelbar mit traditionellen Programmierpraktiken zu befassen. Dies kann dazu beitragen, die Lernkurve erheblich zu verkürzen und die Entwicklung maßgeschneiderter Lösungen zu fördern.

Dennoch sind auch KI-basierte Coding-Assistenten nicht ohne Einschränkungen zu betrachten. Die Qualität des generierten Codes variiert und erfordert eine sorgfältige Überprüfung. Zudem sind grundlegende Programmierkenntnisse weiterhin essenziell, um die Vorschläge der Assistenten effektiv zu nutzen und potenzielle Fehler zu erkennen. Herausforderungen wie das begrenzte Kontextverständnis und die Abhängigkeit von Trainingsdaten unterstreichen die Notwendigkeit eines kritischen Umgangs mit diesen Werkzeugen.

Daher ist für eine erfolgreiche Integration von GPT-basierten Coding-Assistenten in die Ingenieurpraxis ein integrativer Ansatz erforderlich, der verschiedene Aspekte berücksichtigt. Ein solcher Ansatz beinhaltet Schulungen, welche die grundlegenden Programmierfähigkeiten stärken, sowie die Förderung eines kritischen Bewusstseins für die Grenzen und Risiken der KI-Tools. Die Kombination der Stärken von KI-Assistenten mit fundierten Programmierkenntnissen ermöglicht eine bessere Vorbereitung von Ingenieuren auf die Herausforderungen der digitalen Transformation.

Im Rahmen zukünftiger Forschungsarbeiten sollte die Effektivität von GPT-basierten Coding-Assistenten in realen industriellen Anwendungen evaluiert werden. Darauf aufbauend sind Strategien zu entwickeln, um ihr Kontextverständnis und ihre Zuverlässigkeit zu verbessern. Zudem sind Bildungsprogramme anzupassen, um die nächste Generation von Ingenieurinnen und Ingenieuren mit

den notwendigen Fähigkeiten auszustatten, um sowohl traditionelle Programmiermethoden als auch moderne KI-Werkzeuge effektiv nutzen zu können.

Literatur

1. Gabriel, T.; Falkowski, J.; Graunke, R. et al.: Künstliche Intelligenz und industrielle Arbeit – Perspektiven und Gestaltungsoptionen. In: Forschungsbeirat der Plattform Industrie 4.0/acatech – Deutsche Akademie der Technikwissenschaften, 2024 DOI:10.48669/FB40_2024-1
2. Schmidt, C.; Suchy, O.; Stich, A. et al.: KI für die Fachkräftesicherung nutzen: Lösungsansätze für Automatisierung, Teilhabe und Wissenstransfer. Lernende Systeme – Die Plattform für Künstliche Intelligenz, 2024 DOI:10.48669/PLS_2024-2
3. acatech (Hrsg.): Kompetenzen für die Industrie 4.0: Qualifizierungsbedarfe und Lösungsansätze. acatech POSITION. 2016
4. Chen, M.; Tworek, J.; Jun, H. et al.: Evaluating Large Language Models Trained on Code. 2021 DOI:10.48550/arXiv.2107.03374
5. Zhao, Z.; Sun, J.; Cai, C.-H.; Wei, Z.: Code Generation Using Self-Interactive Assistant. In: 2024 IEEE 48th Annual Computers, Software, and Applications Conference (COMPSAC), IEEE. 2024, S. 2347–2352 DOI:10.1109/COMPSAC61105.2024.00377
6. Finnie-Ansley, J.; Denny, P.; Becker, B. A. et al.: The Robots Are Coming: Exploring the Implications of OpenAI Codex on Introductory Programming. In: Proceedings of the 24th Australasian Computing Education Conference, Virtual Event Australia: ACM 2022, S. 10–19 DOI:10.1145/3511861.3511863
7. Vaithilingam, P. M.; Bianculli, D.; Pastore, F.: Expectations, Outcomes, and Challenges of AI-assisted Code Generation: An Empirical Study of GitHub Copilot. In: Proceedings of the 44th International Conference on Software Engineering (ICSE), 2022, S. 577–588 DOI:10.1145/3510003.3510197
8. Nguyen, A.; Nadi, S.; Robillard, M.: A Study of Vulnerabilities in AI-generated Code Recommendations. In: Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis, 2022, S. 20–30 DOI:10.1145/3533767.3534385.
9. Bragança, A.; Azevedo, I.; Bettencourt, N. et al.: Towards Supporting SPL Engineering in Low-code Platforms Using a DSL Approach. In: Proceedings of the 20th ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences, ACM 2021, S. 16–28 DOI:10.1145/3486609.3487196
10. Zhang, Z.; Gray, J.: Test Case Expression in a Low-Code Development Platform. In:

- Proceedings of the 2024 ACM Southeast Conference on ZZZ, ACM 2024, S. 193–198
DOI:10.1145/3603287.3651211
11. Alfonso, I.; Conrardy, A.; Sulejmani, A. et al.: Building BESSER: An Open-Source Low-Code Platform. In: Van Der Aa, H.; Bork, D.; Schmidt, R.; Sturm, A. (eds.): Enterprise, Business-Process and Information Systems Modeling. Lecture Notes in Business Information Processing, Bd. 511., Springer Nature, Cham 2024, S. 203–212
DOI:10.1007/978-3-031-61007-3_16
 12. Kaur, K.; Sharma, S.; Kahlon, K.: A Middleware for Polyglot Persistence and Data Portability of Big Data PaaS Cloud Applications. Computer, Materials & Continua 65 (2020) 2, S. 1625–1647
DOI:10.32604/cmc.2020.011535
 13. Pearce, H.; Ahmad, B.; Tan, B. et al.: Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions. Symposium on Security and Privacy, IEEE 2022
DOI:10.1109/SP46214.2022.9833571
 14. Clark, A.; Igbokwe, D.; Ross, S.; Zibran, M. F.: A Quantitative Analysis of Quality and Consistency in AI-generated Code. In: 7th International Conference on Software and System Engineering (ICoSSE), IEEE 2024, S. 37–41
DOI:10.1109/ICoSSE62619.2024.00014
 15. Corso, V.; Mariani, L.; Micucci, D.; Riganelli, O.: Generating Java Methods: An Empirical Assessment of Four AI-Based Code Assistants. In: Proceedings of the 32nd IEEE/ACM International Conference on Program Comprehension, ACM. 2024, S. 13–23
DOI:10.1145/3643916.3644402
 16. Pinto, G.; De Souza, C.; Rocha, T. et al.: Developer Experiences with a Contextualized AI Coding Assistant: Usability, Expectations, and Outcomes. In: Proceedings of the IEEE/ACM 3rd International Conference on AI Engineering – Software Engineering for AI, ACM 2024, S. 81–91
DOI:10.1145/3644815.3644949
 17. Cotroneo, D.; Improta, C.; Liguori, P.; Natella, R.: Vulnerabilities in AI Code Generators: Exploring Targeted Data Poisoning Attacks. In: Proceedings of the 32nd IEEE/ACM International Conference on Program Comprehension, ACM 2024, S. 280–292
DOI:10.1145/3643916.3644416
 18. Oh, S.; Lee, K.; Park, S. et al.: Poisoned ChatGPT Finds Work for Idle Hands: Exploring Developers' Coding Practices with Insecure Suggestions from Poisoned AI Models. In: IEEE Symposium on Security and Privacy (SP), IEEE 2024, S. 1141–1159
DOI:10.1109/SP54263.2024.00046
 19. Bakharia, A.; Abdi, S.: Shaping Programming and Data Science Education: Insights from GenAI Technical Book Trends. In: IEEE International Conference on Advanced Learning Technologies (ICALT), IEEE 2024, S. 116–120
DOI:10.1109/ICALT61570.2024.00040
 20. Prather, J.; Reeves, B.; Leinonen, J. et al.: The Widening Gap: The Benefits and Harms of Generative AI for Novice Programmers. 2024
DOI:10.48550/ARXIV.2405.17739

Die Autoren dieses Beitrags

Patrick Flore, Jun.-Prof. Dr.-Ing., ist seit 2023 Juniorprofessur für Data Science in Production Engineering an der RPTU in Kaiserslautern und forscht zu den Themen Transfer Lernen, Foundation Models und Generative Methoden zur Datensynthese in der Produktion.

Marco Hussong, M.Sc., ist seit 2021 wissenschaftlicher Mitarbeiter am Lehrstuhl für Fertigungstechnik und Betriebsorganisation (FBK) der RPTU in Kaiserslautern und forscht im Bereich Produktionssysteme zu den Themen digitale Technologien in der Produktion und Nachhaltigkeit in der Produktion.

Peter M. Simon, M.Sc., ist seit 2021 wissenschaftlicher Mitarbeiter am Lehrstuhl für Fertigungstechnik und Betriebsorganisation (FBK) der RPTU in Kaiserslautern und forscht im Bereich Produktionssysteme zu den Themen digitale Technologien in der Produktion.

Abstract

Bridging the Programming Divide – Democratizing Programming skills through GPT Assistants in Manufacturing. Digitalization and AI are transforming the industry – but many engineers face a challenge: a lack of programming skills. How can GPT-based assistants like GitHub Copilot help? This article examines their strengths, compares them to no-code and low-code solutions, and explores challenges in integrating them into daily work. It also highlights the importance of algorithmic problem-solving skills in technical practice.

Förderhinweis

Die Autoren danken dem Ministerium für Wirtschaft, Verkehr, Landwirtschaft und Weinbau vom Land Rheinland-Pfalz für die finanzielle Unterstützung im Rahmen des Forschungsvorhabens „KI4KMU-RLP - Erforschung und Transfer nachhaltiger KI-Innovationen für produzierende KMU in Rheinland-Pfalz“.


Schlüsselwörter

GPT-basierte Coding-Assistenten No-Code-/Low-Code-Plattformen, Künstliche Intelligenz in der Software-Entwicklung, Digitale Transformation

Keywords

GPT-based Coding Assistants, No-Code/Low-Code Platforms, Artificial Intelligence in Software Development, Digital Transformation

Bibliography

DOI:10.1515/zwf-2024-0130
ZWF 120 (2025) Special Issue; page 147 – 151
Open Access. © 2025 bei den Autoren, publiziert von De Gruyter. 
Dieses Werk ist lizenziert unter der Creative Commons Namensnennung 4.0 International Lizenz.
ISSN 0947-0085 · e-ISSN 2511-0896