So programmiert es sich mit Sprach-KI

Sprach-KI soll Programmierern Routineaufgaben abnehmen und Codezeilen beim Schreiben sinnvoll ergänzen. Am Beispiel von GitHub Copilot lässt sich zeigen, wie sich diese Systeme einsetzen lassen und was dabei zu beachten ist. Denn noch darf man sich nicht blind auf die Hilfe verlassen.

Von Philipp Braunhart

-tract

- Der KI-Assistent Copilot von GitHub soll beim Entwickeln helfen und repetitive Standardaufgaben automatisieren.
- Dafür lässt sich der Assistent per Plug-in in gängige IDEs einbinden.
- Copilot berücksichtigt die aktuelle Eingabe und die umliegenden Zeilen, um Entwicklern passende Codevorschläge zu bieten.
- Der Einsatz klappt am besten, wenn man sich das Programm als einen zusätzlichen Entwickler vorstellt, der helfen soll und dementsprechend Informationen benötigt.

„Die neue, heißeste Programmiersprache ist Englisch“, das meinte Andrej Karpathy, der ehemalige Head of AI von Tesla, Anfang des Jahres auf Twitter. Dass er damit nicht ganz falschliegt, zeigt eine Vielzahl von aktuellen Programmierassistenten, die auf generativer künstlicher Intelligenz basieren. Einer davon ist GitHub Copilot, der im Kontext des aktuellen Inhalts meist sehr genau versteht, was gerade gewünscht ist, und sehr selten komplett danebenliegt. Hinter Copilot steckt das KI-Modell Codex von OpenAI, das natürliche Sprache und Code verstehen und generieren kann.

Technischer Hintergrund

GitHub Copilot basiert auf Codex, einer weitertrainierten Variante des Large Language Models GPT-3 von OpenAI. Das Modell verwendet die Transformerarchitektur. Durch mehrere Terabyte an Textdaten unter anderem aus den Archiven von Common Crawl und WebText2, Büchern und der englischen Wikipedia hat OpenAI das Modell auf das Verstehen von Sprache trainiert. Das Feintuning von Codex basiert auf Milliarden von Codezeilen von öffentlichen GitHub-Repositorien. GitHub Copilot filtert und bewertet die Vorschläge von Codex, bevor es diese an die Nutzer sendet.

Copilot erzeugt sowohl Code als auch Kommentare, wobei das Programm den Kontext des zu erzeugenden Codes zur Vorhersage verwendet. Dies ist der Inhalt der Datei mit Fokus auf die aktuelle Zeile und die darüberliegenden. Die Vorschläge können innerhalb einer Zeile sein, eine komplette oder sogar mehrere Zeilen umfassen. Damit lässt sich das Programmieren stark beschleunigen.

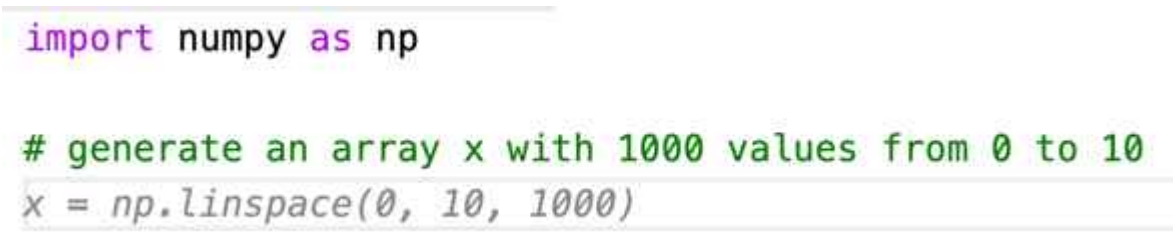
Die Integration in verschiedene IDEs erfolgt durch Plug-ins. Copilot unterstützt nahezu sämtliche Programmiersprachen, am besten sind die Ergebnisse für Python, JavaScript, TypeScript, Ruby, Go, C#, C++ und Java. Auch Sprachen wie SQL, HTML und YAML beherrscht Copilot. Dank dieser Vielfalt ist er nahezu für alle Entwickler eine Hilfe.

Benutzer können sich auf ihre Aufgabe konzentrieren, ohne zwischen der IDE und Webseiten zur Informationssuche wie Stack Overflow zu wechseln. Das kommt gut an: Während der Assistent nach dem Release 27 Prozent der Inhalte von mit ihm editierten Dateien generierte, kommen laut Plattformbetreiber nun 46 Prozent der Files von Copilot, im Fall von Java sogar 61 Prozent, Stand Februar 2023.

Fliegen mit Copilot

Copilot generiert Codevorschläge direkt während des Schreibens. Die Angebote der KI lassen sich per Klick oder Tastendruck annehmen oder ablehnen. Die Schaltflächen über dem Vorschlagsfeld bieten die Möglichkeit, mehrere Vorschläge zu durchsuchen oder sogar ein neues Fenster mit bis zu 10 Entwürfen zu öffnen (siehe Abbildung 1). Eine Alternative besteht darin, Kommentare zu schreiben, um gewünschten Code generieren zu lassen. Dann zeigt Copilot die Codevorschläge in der Zeile unter dem Kommentar an (siehe Abbildung 2).

Einmal in die IDE eingebunden, macht Copilot Vorschläge zum Vervollständigen von angefangenen Codezeilen, hier am Beispiel eines Arrays (Abb. 1).



Copilot lässt sich auch per Kommentar steuern. Hier folgt der KI-Assistent dem Befehl zum Erstellen eines Arrays mit 1000 Werten (Abb. 2).
Das System erstellt nicht nur einzelne Codezeilen: Aus detaillierten Kommentaren oder begonnenen Funktionsdefinitionen mit Input und Output generiert Copilot Codeblöcke. Auch Docstrings erstellt der KI-Assistent. Abbildung 3 zeigt, wie Copilot bereits bei jedem Schritt vom Schreiben der Kommentare bis hin zur Input- und Output-Definition hilft.

```
import pandas as pd
import numpy as np
import plotly.express as px

# generate a dataframe with x from 0 to 10
# and y1 sin, y2 cos
x = np.arange(0, 10, 0.01)
df = pd.DataFrame({
    'x': x,
    'y1': np.sin(x),
    'y2': np.cos(x),
})

# create a function to plot the above dataframe
# with ploty express

def plot(df):
    fig = px.line(df, x='x', y=['y1', 'y2'])
    fig.show()
```

Nichts mehr selbst schreiben: Nur über Kommentare hat Copilot hier den gesamten Code generiert. Dabei lassen sich auch eingebundene Bibliotheken ansteuern (Abb. 3).
Es ist jedoch wichtig zu beachten, dass dabei durchaus Fehler passieren oder das System suboptimalen Code produzieren kann. Daher ist es notwendig, die Vorschläge vor dem Annehmen immer sorgfältig zu überprüfen.

Auch die Qualität der Dokumentation im Code lässt sich so leicht verbessern: Einfach zusätzliche Kommentare und Dokumentation auf Basis des Codekontextes von Copilot generieren lassen. Dabei erkennt und imitiert der Assistent den persönlichen Stil im Code und in den Kommentaren.

Tipps für eine sichere Landung

Ein entscheidender Faktor für die optimale Nutzung von Copilot ist das Bereitstellen von ausreichenden Informationen, um relevante und präzise Vorschläge zu erhalten. Das ist vergleichbar mit einem anderen Programmierer, der auf entsprechende Informationen und Anleitung angewiesen ist. Die wichtigste Frage beim Schreiben von Kommentaren oder Docstrings ist also: Welche Informationen benötigt ein Programmierer, um die Anforderungen zu erfüllen? Je detaillierter die Angaben sind, desto besser kann Copilot weiterhelfen.

Ein weiterer wesentlicher Aspekt ist sauberer und konsistenter Code. Hierbei sollte man gute, klare und präzise Kommentare, Docstrings und aussagekräftige Namen für Funktionen, Klassen, Variablen und Argumente verwenden. Logik und die Intention des Codes sollten leicht zugänglich sein. Copilot kann dabei sogar unterstützen und generiert dadurch anschließend bessere Codevorschläge.

Soll Copilot Funktionen aus Bibliotheken vorschlagen, reicht es meist, Letztere am Anfang der Datei zu importieren. Zusätzlich hilft es, die Bibliothek in einem Kommentar dort zu erwähnen, wo man sie einsetzen will. Bei der Arbeit mit Datenbanken oder Dataframes ist es sinnvoll, das Schema, wie Spaltennamen, Typen und einige Beispiele, als Kommentare bereitzustellen. Dann versteht Copilot die verwendeten Objekte genauer und macht Vorschläge mit den passenden Namen (siehe Abbildung 4).

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

# load data
# DataFrame has columns: ['id', 'name', 'age', 'gender', 'height', 'weight']
pd.read_csv(path)

# histogram
plt.hist(data['height'], bins=10)
```

Informiert man die Sprach-KI per Kommentar über die Namen von Spalten eines Dataframes, dann schlägt das Programm selbstständig mögliche Felder beim Plotten der Daten vor (Abb. 4).
Um Copilot effizient zu nutzen, empfiehlt es sich, Tastenkombinationen in der IDE einzurichten. Geeignete Funktionen für Hotkeys sind: Vorschläge annehmen, ablehnen, nur das nächste Wort des Vorschlags akzeptieren, zwischen verschiedenen Optionen navigieren oder ein separates Fenster mit Vorschlägen öffnen. Dadurch reduziert Copilot die Zeit für repetitive oder mühsame Aufgaben, wie das Schreiben von Unit-Tests, das Behandeln von Exceptions, das Loggen oder das Ausgeben von Informationen.

Tricks für Überflieger

Es gibt auch noch weitere spannende Aspekte an GitHub Copilot jenseits des Vervollständigens von Code. Einer dieser Aspekte ist das Konvertieren von Code in andere Programmiersprachen. Hierdurch lässt sich etwa Legacy-Code leichter portieren (siehe Abbildung 5). Darüber hinaus hilft Copilot auch beim Umwandeln von Code von einem Framework in ein anderes. Das beschleunigt das Konvertieren zwischen verschiedenen Frameworks (siehe Abbildung 6).

```
# Convert this from Python to R
# Python version

import numpy as np
import matplotlib.pyplot as plt

x = np.random.randn(1000)
plt.hist(x, bins=20)
plt.show()

# End

# R version

x <- rnorm(1000)      # <- generated by Copilot
hist(x, breaks=20)    # <- generated by Copilot
```

Will man seinen Code von einer Programmiersprache in eine andere übersetzen, kann eine Sprach-KI helfen. Das Beispiel zeigt das Erstellen und Plotten von 1000 Beispieldaten in Python und in R (Abb. 5).

```
# Rewrite this to use plotly express instead of matplotlib

import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 2 * np.pi, 100)
y = np.sin(x)

plt.plot(x, y)
plt.show()

# plotly express version:

import numpy as np                      # <- generated by Copilot
import plotly.express as px             # <- generated by Copilot

x = np.linspace(0, 2 * np.pi, 100)      # <- generated by Copilot
y = np.sin(x)                           # <- generated by Copilot

fig = px.line(x=x, y=y)                 # <- generated by Copilot
fig.show()                              # <- generated by Copilot
```

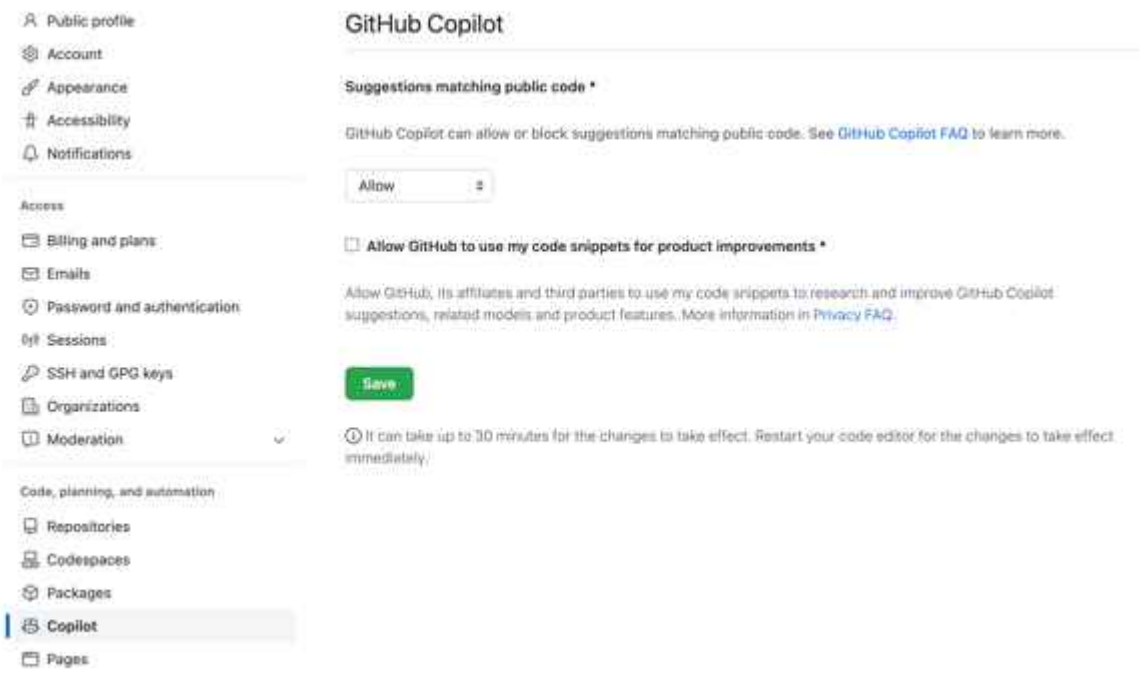
Ähnlich wie beim Übersetzen von Code in unterschiedliche Programmiersprachen dolmetscht Copilot auch zwischen verschiedenen Frameworks und wie hier verschiedenen Bibliotheken einer Sprache (Abb. 6).
Wenn es darum geht, neue Frameworks oder Bibliotheken zu lernen, ist Copilot eine große Hilfe. Vorschläge als Kommentar, basierend auf dem Wunsch des Users, sind ein guter Ausgangspunkt. Damit können Entwickler ihr Wissen schnell erweitern und erfolgreich anwenden.

Zuletzt kann man Copilot auch für das schnelle Generieren von Daten einsetzen. Etwa wenn Daten für Tests nötig sind, reicht die Angabe von ein paar Beispielen und Copilot generiert automatisch zahlreiche weitere Daten.

Turbulenzen

Trotz seines Nutzens weist das Tool jedoch auch Grenzen und potenzielle Risiken auf, die Anwender berücksichtigen sollten. So kann es geschehen, dass Copilot exakte Kopien von Code aus

öffentlichen Repositorys oder Quellen vorschlägt. Hierfür lässt sich eine Filteroption auf der Einstellungsseite aktivieren (siehe Abbildung 7). Dort sieht man auch, dass GitHub den Code von Anwendern standardmäßig für das Retrainieren der KI-Modelle hinter Copilot nutzt und somit potenziell mit anderen Nutzern teilt. Auch dies lässt sich in den Einstellungen deaktivieren.



Zwei besonders wichtige Optionen finden sich im Menü von Copilot. Einerseits lässt sich verhindern, dass der Assistent Code aus öffentlichen Repositorien exakt repliziert. Andererseits kann man per Checkbox verhindern, dass GitHub den gescannten und erzeugten Code zum Trainieren des Modells verwendet und man so möglicherweise Betriebsgeheimnisse verrät (Abb. 7).
Durch das Teilen von eigenem Code können Risiken für sensible Daten entstehen. GitHub versichert, dass die hohen Standards für Datenschutz und Datensicherheit auf ihrer Plattform eingehalten werden und der Code nur zum Erstellen von Vorschlägen genutzt wird. Noch mehr Kontrolle geben Self-Hosting-Alternativen für KI-Codeassistenten (siehe unten).

Generell ist der rechtliche Status der Trainingsdaten von Copilots KI-Modellen noch nicht komplett geklärt und Inhalt einiger aktuell laufender Klagen. OpenAI hat zahlreiche

GitHub-Repositorys, die unter Copyleft-Lizenzen wie der GPL stehen, für das Training verwendet. Copilots Modell weist nicht auf die Lizenzen des eingebauten Codes hin und ist selbst nicht Open Source. Hier bleibt die Rechtsprechung abzuwarten.

Das momentan limitierte Kontextfenster ist eine andere Einschränkung. Der Assistent sollte eigentlich lediglich den Inhalt der aktuell bearbeiteten Datei berücksichtigen. Teilweise entsteht bei der Arbeit allerdings der Eindruck, dass Copilot auch andere offene Dateien heranzieht. Allerdings verwendet das System nicht die gesamte Codebasis eines Projekts.

Unsicherer Code und Schwachstellen stellen ebenfalls eine Herausforderung dar. Eine aktuelle Forschungsarbeit deutet darauf hin, dass Nutzer von Tools wie Copilot dazu tendieren, Code mit Schwachstellen leichter zu akzeptieren, sich aber gleichzeitig in Sicherheit wiegen (siehe ix.de/zxwq). Mit einem Copilot-Update im Februar 2023 führte GitHub ein Feature ein, das potenzielle Schwachstellen in den Vorschlägen scannt und automatisch unterdrückt. Das System filtert dabei typische Schwachstellen wie hartcodierte Credentials, SQL- oder Pfadinjektionen.

Einige Probleme sind damit sicherlich behoben, aber natürlich findet das Programm dadurch nicht alle Schwachstellen. Es bedarf also weiterhin großer Aufmerksamkeit, gerade bei sicherheitsrelevanten Anwendungen. Zuletzt sei erwähnt, dass Copilot am besten auf Englisch arbeitet. Arbeiten Anwender mit anderen Sprachen, kann das zu Einschränkungen führen.

Alternative Reisemöglichkeiten

Zu Copilot gibt es diverse Alternativen, darunter sowohl kommerzielle als auch Open Source. AWS bietet beispielsweise Code Whisperer an, Salesforce mit CodeGen ein quelloffenes Projekt, das sich mit Fauxpilot selbst hosten lässt.

Weitere Optionen sind Tabnine und Replit Ghostwriter. Für JupyterLab, Colab-Notebooks oder BigQuery ist zudem CodeSquire als Browsererweiterung verfügbar. Der Markt entwickelt sich stetig und so kommen und gehen die Alternativen. Unter allen Optionen scheint GitHub Copilot derzeit am ausgereiftesten und zuverlässigsten zu sein.

Was klar ist: Copilot ist mehr als nur ein einfaches Autocomplete-Werkzeug. Es ist der Anfang eines größeren Wandels im Programmieren, der darauf abzielt, mehr Aspekte des Entwicklungsprozesses zu automatisieren.

Die Fähigkeit, Code aus Kommentaren und Beispielen innerhalb der eigenen IDE zu generieren, reduziert Ablenkungen, beschleunigt das Programmieren und verbessert die Qualität der Dokumentation. Für den richtigen Einsatz und gute Vorschläge sollten Entwickler im Kopf behalten, dass das System die gleichen Informationen benötigt wie ein Kollege, den man um Rat bittet. Forscher bei Google unterstreichen die Vorteile von Programmierassistenten. In einer Untersuchung mit der Inhouse-KI will man herausgefunden haben, dass sich die Produktivität mithilfe des Tools messbar erhöhen lässt. (pst@ix.de)

1. Quellen
2. Weitere Informationen zu Copilot und dem Programmieren mit KI-Assistenten finden sich unter ix.de/zxwq.

Philipp Braunhart

ist Entwickler und Berater im Bereich Machine Learning und Mitgründer der ingenio ai GmbH. Er programmiert, entwirft und

baut KI-Lösungen für den direkten Einsatz in den Bereichen Bio-, Agrar- und Medizintechnologie.