



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Willkommen

VOM COPY-PASTE ZUM
DURCHDACHTEN PROMPT

Heutige Agenda

1. **Feedback** zu Ihrer Stack Overflow Challenge-Erfahrung
2. **Drei Ansätze** für dieselbe Challenge — Copy-Paste → Strukturiert → Professionell
3. **Das KI-Tagebuch** — Prompts als Engineering-Artefakte dokumentieren
4. **GitHub Copilot & Claude Code** — zwei Werkzeuge, drei Modi
5. **Challenge Runde 2** — dieselbe Aufgabe, neuer Ansatz

Zentrale Erkenntnis: *Die Art, wie Sie mit einem KI-Werkzeug interagieren, ist wichtiger als die Wahl des Werkzeugs.*

Rückblick: Die Stack Overflow Challenge

In Vorlesung 1 haben Sie diese Aufgabe erhalten:

Ihre Mission: Erstellen Sie den Stack Overflow-Plot mit Python und KI-Werkzeugen nach.

- Finden Sie die Daten **programmatisch**
- Reproduzieren Sie das Diagramm: monatliche Fragenzahlen + gleitender Durchschnitt
- **Erweitern Sie es:** Überlagern Sie Markierungen für wichtige KI-Durchbrüche

Vollständige Übung: [Stack Overflow Challenge — Runde 1](#)

Wie ist es gelaufen?

Kurze Handzeichen-Runde:

- Wer hat einen **funktionierenden Plot** erstellt?
- Wer hat **teilweise funktionierenden** Code erhalten?
- Wer ist beim **Datenzugriff hängengeblieben**?
- Wer hat **halluzinierte API-Endpunkte** erlebt?
- Wer hat **jede Zeile** des KI-generierten Codes gelesen?

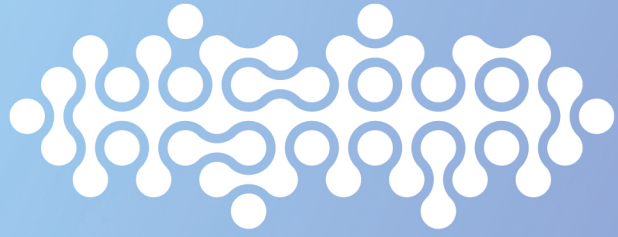
Feedback-Runde

Bitte füllen Sie dieses kurze Feedback-Quiz zu Ihrer Erfahrung aus:

Stack Overflow Challenge — Feedback zum ersten Versuch

Wir werden die aggregierten Ergebnisse gemeinsam betrachten.

Achten Sie auf die Muster — Ihre Schwierigkeiten wurden wahrscheinlich von vielen Kommiliton:innen geteilt.



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Drei Ansätze für die Challenge

1

Ansatz 1: Das Copy-Paste

Was Sie tun: Aufgabenbeschreibung nehmen und in den KI-Chat einfügen. Fertig.

Your mission: Recreate the Stack Overflow plot using Python + AI tools.

- Find the data programmatically (Stack Overflow Data Explorer, ...)
- Reproduce the chart: monthly question counts + rolling average
- Extend it: Overlay markers for key AI breakthroughs

Create a Python script that does all of this.

Oder: einen **Screenshot der Folie** direkt in den Chat einfügen.

Warum sind alle Prompts auf Englisch?

LLMs werden überwiegend mit englischsprachigem Text trainiert (~90 %). Forschung zeigt:

- **Englische Prompts liefern besseres Reasoning** — selbst wenn die Frage in einer anderen Sprache gestellt wird ([Shi et al., 2022](#); [Etxaniz et al., 2023](#))
- **„Cross-Lingual-Thought Prompting“** — Reasoning über Englisch zu routen bringt signifikante Verbesserungen ([Huang et al., 2023](#))
- **Interne Repräsentationen sind auf Englisch ausgerichtet** — die mittleren Schichten, in denen Reasoning stattfindet, arbeiten in einem englischzentrierten Raum ([Wendler et al., 2024](#))

Übersetzt das Modell Ihren deutschen Prompt intern? Keine wörtliche Übersetzung — aber die stärksten Reasoning-Pfade wurden durch englische Daten geformt.

Best Practice: Prompten Sie auf Englisch, fordern Sie die Ausgabe in der gewünschten Sprache an. Das Modell generiert deutsche Tokens direkt — kein versteckter

Übersetzungsschritt.

Was passiert

Die KI generiert ein vollständiges 50-100-Zeilen-Skript. Sieht beeindruckend aus.

Aber dann:

- **Halluzinierte API-Endpunkte** — plausible URLs, die 404 zurückgeben
- **Hartcodierte Fake-Daten** — Diagramm sieht toll aus, Zahlen sind erfunden
- **Monolithische Struktur** — alles in einer Funktion, nicht testbar
- **Fehlende Abhängigkeiten** — `ModuleNotFoundError` beim ersten Start

Was folgt: 10–15 Runden von „*Das funktioniert nicht, weil...*“

Das 80/20-Problem

Die ersten 80 %:

- KI bringt Sie in Minuten zu „sieht aus, als würde es funktionieren“
- Die anfängliche Geschwindigkeit ist verführerisch

Die letzten 20 %:

- Debugging, Datenprobleme, subtile Bugs
- Dauert Stunden, nicht Minuten
- Jede Korrektur riskiert, andere Teile zu beschädigen

Halluzinations-Angriffsfläche: Ein breiter Prompt ohne Einschränkungen gibt der KI maximale Freiheit, plausibel klingende, aber falsche Inhalte zu generieren.

Haben Sie den Code verstanden?

Ehrliche Frage: Als die KI 80 Zeilen Code in Sekunden generiert hat...

- Haben Sie ihn **gelesen**?
- Könnten Sie **erklären**, warum er genau diese API-Parameter verwendet?
- Könnten Sie das **Datums-Parsing reparieren**, wenn es auf neue Art bricht?

Code zu verwenden, den Sie nicht verstehen, macht ihn **deutlich schwieriger zu warten** — selbst mit KI-Werkzeugen für zukünftige Änderungen.

Das ist der entscheidende Punkt. Keine Regel — ein **Bewusstsein**.

Ansatz 2: Die strukturierte Zerlegung

Was Sie tun: 5–10 Minuten nachdenken, **bevor** Sie die KI anfassen.

Verschiedene Anliegen identifizieren:

1. **Datenrecherche** — Woher kommen die Daten?
2. **Datenzugriff** — Abrufen und Cachen
3. **Visualisierung** — Plotten mit Formatierung
4. **Meilenstein-Daten** — Die 6 KI-Durchbruchsdaten
5. **Integration** — Alles zusammenführen

Dann **mehrere fokussierte Prompts** statt eines großen schreiben.

Erfahrung formt Ihre Prompts

Die folgenden Beispiel-Prompts spiegeln jemanden mit API- und Python-Erfahrung wider. **Ihre Prompts werden anders aussehen — und das ist völlig in Ordnung.**

Erfahrene Entwicklerin:

"Fetch monthly SO counts using SEDE, return as pandas DataFrame with columns year_month, question_count"

Neuere Entwicklerin:

"I need monthly Stack Overflow question counts from 2008-2024. How can I get this data?"

Die **Strategie** zählt (zerlegen, erst recherchieren), nicht die spezifischen Begriffe.

Prompt 1 — Recherche

I need to get the monthly count of questions posted on Stack Overflow from 2008 to 2024. What are my options?

Consider: Stack Exchange API, SEDE, data dumps, web scraping, pre-existing datasets.

For each: pros, cons, rate limits, access requirements.

Dieser Prompt fragt nach **Recherche**, nicht nach Code.

Sie nutzen die KI als **Analystin**, bevor Sie sie als Programmiererin einsetzen.

Prompt 2 — Datenzugriffsmodul

Write a Python module ``data_fetcher.py`` that fetches monthly SO question counts using SEDE.

Requirements:

- Return data as a pandas DataFrame
- Cache results locally as CSV
- Handle network errors with retry logic

Fokussierte Aufgabe mit einer Verantwortlichkeit. Präzise Anforderungen lassen weniger Raum für Halluzinationen.

Übermäßige Vorgaben können auch schaden — die KI findet vielleicht einen besseren Ansatz, wenn Sie etwas Freiheit lassen.

Prompt 4 — Meilenstein-Daten

Die Aufgabe sagt: „Überlagern Sie Markierungen für wichtige KI-Durchbrüche aus der Zeitleiste.“

Das sind **bekannte Daten** aus der Kurs-Zeitleiste:

- Transformer (Juni 2017) · ChatGPT (Nov 2022) · GPT-4 (März 2023)
- Cursor & KI-IDEs (Aug 2023) · Claude Code (Feb 2025) · Copilot Agent Mode (Feb 2025)

Bei nur 6 Meilensteinen könnten Sie sie einfach **manuell eintippen**. Das ist eine valide Engineering-Entscheidung.

Nicht alles muss KI-generiert werden. Zu wissen, wann man die KI *nicht* einsetzt, ist Teil der Kompetenz.

Was bei Ansatz 2 anders ist

- Jedes Modul ist **eigenständig** und unabhängig testbar
- Halluzinationen sind **isoliert** — ein fehlerhafter API-Aufruf ist in `data_fetcher.py`, nicht überall
- **Recherche vor Implementierung** liefert konsistent bessere Ergebnisse

Zentrale Erkenntnis: *Prompting ist Design.* Wie Sie Prompts zerlegen, spiegelt wider, wie Sie Software zerlegen. Das ist kein Zufall.

Könnte das ein einziger Prompt sein?

```
I need a Python project that visualizes monthly SO question counts with AI milestone markers.
```

```
Before writing code, first:
```

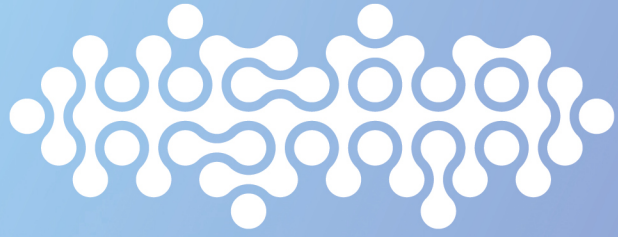
1. Research available data sources
2. Design a modular architecture
3. Define each module's interface

```
Then implement each module one at a time.
```

Das codiert den **Denkprozess** als Anweisungen. Ein Prompt, aber strukturiert.

Was Sie gewinnen: Bequemlichkeit. **Was Sie verlieren:** Kontrolle über Designentscheidungen.

Vorschau: Das ist ein Sprungbrett zu **agentischen Workflows** (Akt III).



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Warum KI-Interaktionen dokumentieren?

2

Das Problem mit der Chat-Historie

Die eingebaute Chat-Historie (Copilot, ChatGPT) hat **erhebliche Lücken**:

- **Keine Artefakt-Zuordnung** — welcher Prompt führte zu welcher Code-Änderung?
- **Kein Iterations-Tracking** — wie viele Hin-und-her-Zyklen brauchte eine Aufgabe?
- **Kein Intent-Tracking** — was Sie *tun wollten* vs. was Sie *gefragt haben*
- **Kein werkzeugübergreifender Vergleich** — Historien an verschiedenen Orten, in verschiedenen Formaten
- **Fragile Speicherung** — Sitzungen können verloren gehen, wenn VS Code neu startet

Warum Tracking wichtig ist

Selbstreflexion. Sie erlernen eine neue Fähigkeit. Verbesserung erfordert bewusste Reflexion. Ohne Aufzeichnungen basiert das auf vager Erinnerung.

Zukünftige Kursarbeit. In Akt III werden Sie Ihre eigenen Prompting-Strategien analysieren. Dafür brauchen Sie *Daten* — Prompt-Anzahlen, Halluzinationsmuster, Iterationstrends. Das Tagebuch ist Ihr Datensatz.

Professionelle Evidenz. Am Ende dieses Kurses ist Ihr Tagebuch ein Portfolio darüber, *wie* Sie mit KI-Werkzeugen arbeiten — ein wertvolleres Signal als Code allein.

Ansatz 3: Der Professionelle

Bevor Sie überhaupt prompten, entscheiden Sie sich für einen **professionellen Entwicklungsprozess**:

1. Projekt mit ordentlicher Paketierung aufsetzen — **Git-Repo von Anfang an**
2. Jeden bedeutsamen Meilenstein separat committen
3. Anforderungen **als Dokument** definieren, bevor Code geschrieben wird
4. **Tests vor der Implementierung** schreiben
5. Infrastruktur-Tooling wählen (`uv` , `gh`)
6. Jede KI-Interaktion in einem `diary/` **-Ordner** dokumentieren

Das bettet den **Software Development Lifecycle** in Ihre Prompting-Strategie ein.

Prompt 0 — Projektaufbau (Teil 1)

Create a Python project called so-challenge using uv for dependency management. Separate data collection (data_fetcher.py) from visualization (plotter.py), with a module for milestones (milestones.py).

Set up pytest with test files for each module. Manage all dependencies through pyproject.toml. Include a README.md.

Create a diary/ folder for tracking AI interactions.

Das ist der **Scaffolding**-Teil — welche Dateien und Strukturen erstellt werden. Aber der Prompt geht weiter...

Prompt 0 — Tagebuchformat (Teil 2)

For every prompt cycle, save the interaction record to diary/ as a numbered markdown file. Use this format:

```
# NNN — Short Title
**Date**: YYYY-MM-DD  **Tool**: [name]  **Model**: [name]
**Iterations**: [number]
## Prompt
**YYYY-MM-DD HH:MM**
[The full prompt text as given by the user.]
```

Save this setup prompt as diary/001-project-setup.md.

Sie müssen das Format definieren — sonst erfindet die KI jedes Mal ein anderes.

Prompt 0 — Einschränkungen (Teil 3)

Do NOT implement any logic yet – just create the project structure, configuration, and empty module files with docstrings describing their purpose.

Initialize a git repository and create an initial commit.
Then create a GitHub repository using the gh CLI and push.

Drei kritische Elemente:

- „**Do NOT implement**“ — kontrolliert, was die KI *nicht* tun soll
- **Git von Anfang an** — Scaffolding bekommt eigenen Commit vor jeglichem Code
- **Absicht statt Layout** — beschreibt *was Sie wollen*, keinen ASCII-Verzeichnisbaum

Prompt 1 — Anforderungsdokument

Write a brief requirements specification for this project and save it to docs/requirements.md.

Include:

- Functional requirements (data source, date range, plot type)
- Non-functional requirements (caching, error handling)
- Acceptance criteria for each requirement

Save the diary entry and commit everything together with a meaningful commit message.

Das produziert ein **Dokument, keinen Code** — und committet es sofort mit seinem Tagebucheintrag.

Prompt 2 — Tests zuerst (Red Phase)

Write pytest tests for `data_fetcher.py`. The module should:

- Fetch monthly SO question counts (2008-2024)
- Return a pandas DataFrame
- Cache results locally as CSV
- Handle network errors gracefully

Write the tests BEFORE the implementation. Use mocks for network calls.

Save the diary entry and commit with a meaningful commit message.

Diese Tests werden **fehlschlagen** — das ist der Sinn. Sie definieren das erwartete Verhalten *vor* der Implementierung.

Prompt 3 — Implementierung (Green Phase)

```
Implement data_fetcher.py to pass the tests. Do not modify  
the tests. Run the tests to confirm they pass.
```

```
Save the diary entry and commit with a meaningful commit  
message describing what was implemented.
```

Kein Einfügen des Testcodes nötig — Sie sind in derselben Sitzung, die KI hat den Kontext bereits.

„Proper commit message“ — die KI leitet sie aus den Änderungen ab. In einem realen Projekt können Commit-Vorlagen und Konventionen in Projektanweisungsdateien codiert werden.

Wiederholen: Red → Green → Commit

Wenden Sie dasselbe Muster für jedes Modul an:

plotter.py : Tests → Implementierung → Commit

milestones.py : Tests → Implementierung → Commit

main.py : Integration → Commit

Jeder Zyklus produziert zwei Commits: erst Tests, dann Implementierung. Jeder Commit enthält seinen zugehörigen Tagebucheintrag.

Oder mit einem **agentischen Werkzeug** wie Claude Code können Sie den Zyklus komprimieren:

"Implement data_fetcher with tests. If tests pass, commit and move to plotter."

Die resultierende Git-Historie

```
* f4e2a1b Integrate main.py and update README
* c8d3b7e Implement milestones module
* 91a0f5c Add tests for milestones module
* 7b2e4d9 Implement plotter module
* 3f8c1a6 Add tests for plotter module
* a5d9e2f Implement data_fetcher module
* e1c7b3d Add tests for data_fetcher module
* 6d4f8a1 Add requirements specification
* 9c3e8f1 Initial project scaffolding
```

Jeder Commit erzählt eine Geschichte. Jeder enthält einen Tagebucheintrag im `diary/` - Ordner.

`git show` auf einem beliebigen Commit → Code-Änderungen + der Prompt, der sie hervorgebracht hat.

Was Ansatz 3 anders macht

- **Tests als Halluzinations-Detektoren** — halluzinierter Code schlägt sofort fehl
- **Infrastruktur zählt** — `uv sync` und alles funktioniert, keine „bei mir läuft's"-Probleme
- **SDLC eingebettet ins Prompting** — Anforderungen → Tests → Implementierung → Commit
- **Die KI managt Prozesse, nicht nur Code** — Anforderungsdokumente, Commit-Messages, Tagebucheinträge

Gleicher Erfahrungsvorbehalt: Wenn `uv`, `pytest` und TDD neu für Sie sind, starten Sie einfacher. Das **Kernprinzip** bleibt: Lassen Sie den Engineering-Prozess Ihr Prompting leiten.

Das Wiederholungsproblem

Zwei Ärgernisse bemerkt:

1. **Prompt 0** enthält eine lange Tagebuchformat-Spezifikation
2. **Jeder folgende Prompt** enthält „save the diary entry and commit“

Innerhalb einer Sitzung merkt sich die KI das. **Aber morgen? Neue Sitzung = kein Gedächtnis.**

Das ist nicht nur ein Tagebuchproblem. Denken Sie an alle Konventionen, die Sie wiederholen:

- Commit-Message-Format
- Testing-Anforderungen
- Coding-Standards
- Bibliothekspräferenzen

Projektanweisungsdateien

Textdateien, die die KI **automatisch** zu Beginn jeder Sitzung liest:

Werkzeug	Datei
Claude Code	CLAUDE.md
GitHub Copilot	.github/copilot-instructions.md
Cursor	.cursorrules

Gleiches Konzept, verschiedene Dateinamen. **Konventionen einmal codieren, das Werkzeug wendet sie immer an.**

Wir richten diese in Akt III ein. Vorerst: verstehen Sie, **warum** sie existieren.

CLAUDE.md-Beispiel

```
# Project Instructions
```

```
## AI Diary
```

```
For every prompt cycle, save a diary entry to diary/  
as a numbered markdown file with: date, tool, model,  
iterations, and full prompt text with timestamps.  
Always commit diary entries with the code changes.
```

```
## Conventions
```

- Run tests before committing implementation code
- Use meaningful commit messages derived from changes
- Do not modify existing tests when implementing

Mit dieser Datei erwähnen Sie das Tagebuch in keinem Prompt. **Jede Sitzung weiß es einfach.**

Diskussion: Ihre CLAUDE.md

Wenn Sie eine CLAUDE.md für Ihr Projekt einrichten würden, was würden Sie hineinschreiben?

Denken Sie an:

- Was wiederholen Sie in Ihren Prompts?
- Welche Konventionen befolgt Ihr Team?
- Was sollte die KI *nie* tun?
- Was sollte die KI *immer* tun?

Nehmen Sie sich 2 Minuten. Teilen Sie eine Idee mit dem Kurs.

Vergleich der drei Ansätze

Aspekt	Copy-Paste	Strukturiert	Professionell
Einrichtung	Keine	Minuten Nachdenken	Erheblich
Prompts	1 + Nachbesserungen	4–6 fokussierte	8–12 systematische
Halluzinationsrisiko	Hoch	Mittel	Gering (Tests!)
Debugging	Schwer	Mittel	Leicht
Reproduzierbarkeit	Gering	Mittel	Hoch

Zeitschätzungen sind **erfahrungsabhängig** — das relative Muster gilt, nicht die absoluten Zahlen.

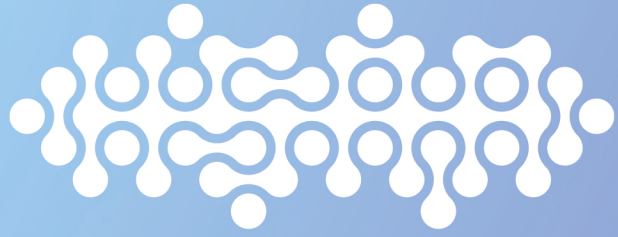
Kernaussagen

Keiner dieser Ansätze ist „falsch“. Sie stellen eine Progression dar.

- Ansatz 1: in Ordnung für schnelle Experimente
- Ansatz 2: angemessen für die meisten Aufgaben
- Ansatz 3: wenn Qualität und Wartbarkeit wichtig sind

Es geht nicht um Prompting-Tricks. Es geht darum, **Software-Engineering-Denken** auf KI-gestützte Entwicklung anzuwenden.

Die KI weiß nicht, was „gut“ für Ihr Projekt bedeutet. Sie codieren Qualität durch Struktur, Anforderungen und Tests.



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Das KI-Tagebuch

3

Tagebucheintrag-Format

```
# 004 – Data Fetcher Implementation

**Date**: 2026-03-23
**Tool**: Claude Code
**Model**: Claude Opus 4.6
**Iterations**: 2

## Prompt

**2026-03-23 10:25**
Implement data_fetcher.py to pass the tests...

**2026-03-23 10:32**
The test test_retries_on_network_error is failing...
```

Vollständige Prompts mit Zeitstempeln. Keine Zusammenfassungen. Rohaufzeichnung der Konversation.

Projektstruktur mit Tagebuch

```
so-challenge/  
├─ src/so_challenge/  
│   ├── data_fetcher.py  
│   ├── plotter.py  
│   └─ milestones.py  
├─ tests/  
├─ docs/requirements.md  
├─ diary/  
│   ├── 001-project-setup.md  
│   ├── 002-requirements.md  
│   ├── 003-data-fetcher-tests.md  
│   ├── 004-data-fetcher-impl.md  
│   └─ ...  
└─ pyproject.toml
```

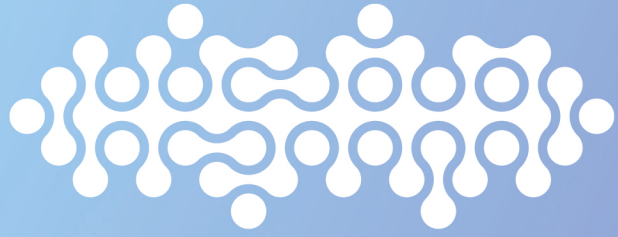
Jeder Tagebucheintrag **zusammen** mit dem Code committet, den er produzierte.

Warum Prompts versionieren?

In einem Produktionsprojekt würden Sie das wahrscheinlich nicht tun. Aber **dies ist ein Kurs über den Aufbau von KI-Kompetenz.**

Prompts sind Engineering-Artefakte. Wenn Sie Ihren Quellcode nicht wegwerfen würden, warum dann die Anweisungen, die ihn erzeugt haben?

- **Sehen, welcher Prompt welchen Code erzeugt hat** — ein `git diff` erzählt die Geschichte
- **Prompts über die Zeit vergleichen** — hat sich Ihr Prompting zwischen Woche 2 und Woche 10 verbessert?
- **Modelle vergleichen** — derselbe Prompt an Copilot vs. Claude, leicht zu diffen
- **Teilen und diskutieren** — Prompts liefern Kontext, den Code allein nicht bieten kann



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

GitHub Copilot & Claude Code

4

Zwei KI-Werkzeuge in VS Code

Im Agent-Modus können **beide Werkzeuge**: Dateien lesen/schreiben, Befehle ausführen, Tests starten, Git-Commits erstellen, `gh` CLI nutzen. Die Kernfähigkeiten sind konvergiert.

Aspekt	GitHub Copilot	Claude Code
Oberfläche	Chat-Panel + Inline-Vervollständigungen	Terminal-Panel + eigenständige CLI
Modelle	GPT-4o, Claude Sonnet, Gemini	Claude Opus, Claude Sonnet
Kontextfenster	128K–200K Tokens	Bis zu 1M Tokens (Opus)
Anweisungsdatei	<code>.github/copilot-instructions.md</code>	<code>CLAUDE.md</code>
Preis für Studierende	Kostenlos (GitHub Education)	Kostenlose Stufe (begrenzt)

Interaktionsmodi

Modus	GitHub Copilot	Claude Code
Ask	Ask — antwortet, keine Bearbeitungen	Ask — antwortet, keine Bearbeitungen
Agent / Edit	Agent — bearbeitet, Sie prüfen	Edit automatically
Plan	Plan über <code>/plan</code>	Plan — erst entwerfen

Die Namen unterscheiden sich, das Konzept ist dasselbe: **Wie viel Autonomie geben Sie der KI?**

- **Ask:** KI ist Beraterin. Sie berät, Sie führen aus.
- **Agent/Edit:** KI ist Kollaborateurin. Sie ändert, Sie prüfen.
- **Plan:** KI ist Architektin. Sie entwirft, Sie genehmigen, dann baut sie.

Wann welches Werkzeug

Beide Werkzeuge können im Agent-Modus fast jede Aufgabe bewältigen. Einige Anfangsbeobachtungen:

- **Copilots Inline-Vervollständigungen** (außerhalb des Agent-Modus) sind ideal für fließende Bearbeitungen — kein Äquivalent in Claude Code
- **Copilots Modellflexibilität** — Wechsel zwischen GPT-4o, Claude Sonnet, Gemini in einer Sitzung
- **Claude Codes Kontextfenster** — bis zu 1M Tokens, Vorteil bei großen Codebasen
- **Claude Codes eigenständige CLI** — funktioniert außerhalb von VS Code, nützlich für Skripting und CI/CD

Probieren Sie beide aus und vergleichen Sie. Dokumentieren Sie Ihre Präferenzen im KI-Tagebuch — genau dafür ist es da.

Berechtigungen konfigurieren

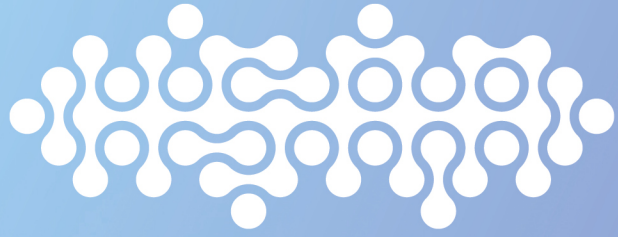
Beide Werkzeuge fragen vor dem Ausführen von Befehlen nach. Für wiederholte Workflows Auto-Genehmigung konfigurieren:

Claude Code — `~/.claude/settings.json`:

```
{ "permissions": { "allow": [
  "Edit", "Write", "Bash(git:*)", "Bash(uv:*)"
] } }
```

Copilot — VS Code Trust-Modell: „Allow“ oder „Always Allow“ pro Befehlsmuster klicken.

Best Practice: Mit Standardeinstellungen beginnen, Berechtigungen hinzufügen, wenn Sie entdecken, was Sie regelmäßig ausführen.



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Challenge Runde 2

5

Stack Overflow Challenge — Runde 2

Dieselbe Aufgabe. Ihr Ansatz sollte nicht derselbe sein.

Verwenden Sie **Ansatz 3** — den professionellen SDLC-Workflow mit integriertem Tagebuch.

Abzugebende Leistungen:

- Ein GitHub-Repo mit sauberer Commit-Historie (Code + Tagebucheinträge pro Commit)
- Ein `diary/`-Ordner mit nummerierten Einträgen
- Der generierte Stack Overflow-Trend-Plot

Referenz-Repo: github.com/l3gj0n/so-challenge

Vollständige Übung: [Runde 2](#)

Fokus auf den Prozess, nicht nur das Ergebnis

Ein **funktionierender Plot** ohne Commit-Historie und ohne Tagebuch ist weniger wert als ein **teilweise funktionierender Plot** mit einem sauberen Git-Log und vollständigen Tagebucheinträgen.

Wir bewerten, **wie** Sie arbeiten, nicht nur **was** Sie produzieren.

Zeitplan: Beginnen Sie während dieser Sitzung, wenn die Zeit es erlaubt. Fertigstellen bis Vorlesung 3.

Feedback nach der Challenge wird gesammelt, nachdem Sie Runde 2 abgeschlossen haben — wir vergleichen Ihre Erfahrung über beide Versuche zu Beginn von Vorlesung 3.



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Fragen?

VIELEN DANK

Ihr Feedback ist willkommen.