◆TECH&TALK · SOLINGEN 09.09.2026 · KÖLN 10.09.2026

# From Prompts to Skills: Advanced GitHub Copilot Workflows

Skills, Memory Bank, Rubber Duck & Bring Your Own Model –
aus dem täglichen Einsatz, mit echten Messwerten

**Dr. Matthias Liebeck**

liebeck.io

# Kurz zu mir: Matthias Liebeck

- Senior Solution Architect in Düsseldorf, Schwerpunkt .NET, C# und Azure
- Software-Entwickler seit 2009, C# seit 2006
- 2009 – 2015: Studium Informatik & Mathematik
- 2018: Promotion in Informatik über Natural Language Processing / KI
- Organisator des Azure Düsseldorf Meetups, Speaker über Azure & agentic coding
- GitHub Copilot Newsletter: ghcp.liebeck.io

Azure Meetup Düsseldorf

MVP Microsoft Most Valuable Professional

Microsoft Certified: Azure Solutions Architect Expert

Microsoft Certified: DevOps Engineer Expert

Microsoft Certified: Azure Developer Associate

Microsoft Certified: Azure Administrator Associate

Microsoft Certified: Azure Data Scientist Associate

GitHub Copilot Certification Program

GitHub Copilot Newsletter

# ✋ Wer weiß, was agentic coding ist?

Du beschreibst das Ziel einer Aufgabe als Prompt in einem Chatfenster. Ein KI-Agent plant die Schritte, liest deinen Code, ändert Dateien, führt Builds und Tests aus und meldet sich, wenn er Feedback braucht. Nicht Autovervollständigung, sondern delegierte Arbeit.

**GitHub Copilot** GitHubs Coding-Agent – in IDE, CLI und eigener App

**Claude Code** Anthropics Coding-Agent – Terminal, IDE-Anbindung und Desktop

**Codex** OpenAIs Coding-Agent – Cloud, CLI und IDE-Extension

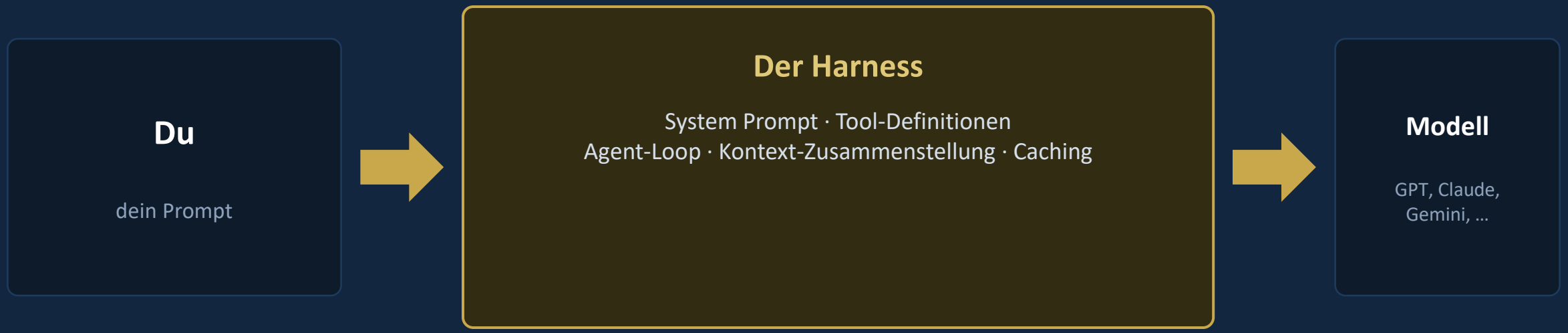
# ✋ Wer nutzt agentic coding?

# ✋ Und womit?

| | |
|---|---|
| GitHub Copilot | Claude Code |
| Codex | etwas anderes |

# ✋ Wer weiß, was ein Harness ist?

Der Harness ist alles, was ein Coding-Agent um das Modell herum baut: System Prompt, Werkzeuge, Agent-Loop und Kontext-Verwaltung. Ihr redet nie direkt mit dem Modell – immer mit dem Harness.




*Was der Harness kostet, sehen wir gleich in Block 01 – mit echten Zahlen.*

◆MEIN ANSPRUCH AN HEUTE

# Was ihr mitnehmen sollt

## Breite statt Tiefe

Bei Interesse gerne Fragen stellen

## Mindestens ein Aha-Moment

Etwas zeigen, das ihr noch nicht kanntet

## Ein konkreter Impuls

Eine Sache, die ihr nächste Woche in euren eigenen Workflow übernehmt, unabhängig vom jeweiligen agentic coding Anbieter

◆ AGENDA

# Heute Abend

**01** **Tokens & Harness**

Was kostet ein „OK“ – und warum?

**02** **Custom Instructions & Custom Agents**

Das Fundament – und eine ehrliche Einordnung

**03** **Agent Skills**

Wissen einmal definieren, überall nutzen

**04** **Memory Bank**

Gedächtnis über Sessions hinweg – in Git statt Cloud

**05** **Rubber Duck & Slash Commands**

Die Ente redet zurück – und ihr frischer Nachfolger HydraFusion

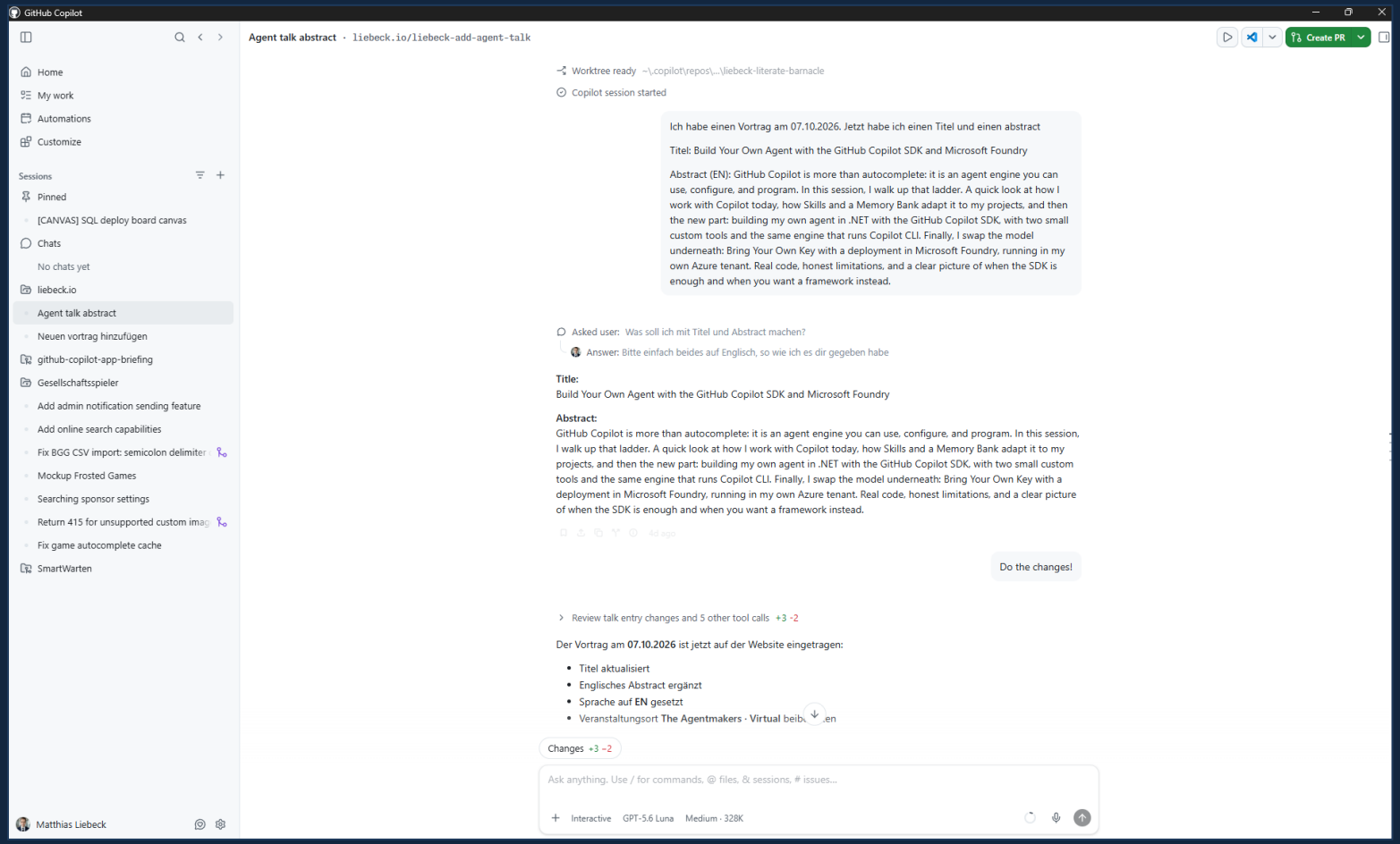
**06** **Bring Your Own Model**

Fünf Wege zu deinem Modell – inkl. OpenRouter & Foundry

COPILOT-SPEZIFISCH

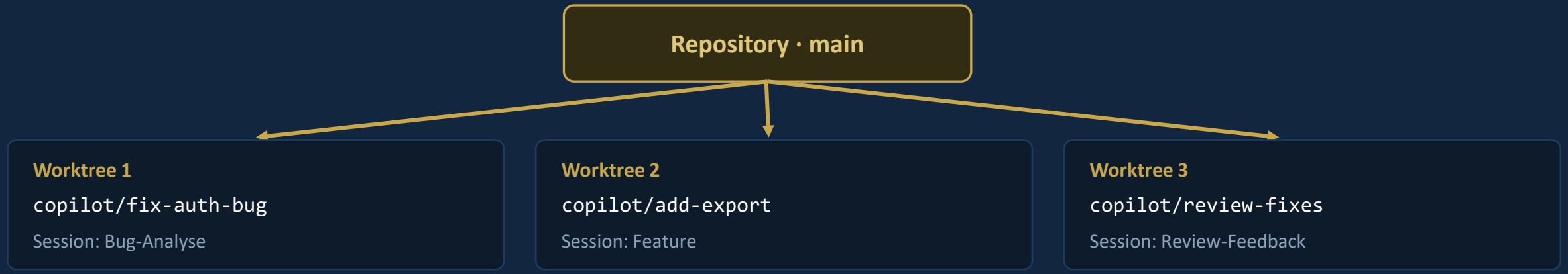
# Meine neue Entwicklungsumgebung

Die GitHub Copilot App ist mein Cockpit für die tägliche Arbeit geworden. Visual Studio öffne ich nur noch, um den Debugger zu starten.



KONZEPT PORTABEL

# Jede Session bekommt ihren eigenen Worktree




- Ein git worktree ist ein echter, isolierter zweiter Checkout des Repos in einem eigenen Ordner
- Parallele Sessions kommen sich nie in die Quere – die App legt Worktrees an und räumt sie wieder auf
- **Bei mir laufen bis zu 5 Sessions parallel – ich wechsle, sobald eine nach Feedback fragt**
- Dasselbe Prinzip gibt es in Claude (Code & Desktop) und bei Codex – wichtig ist, dass ihr wisst, dass es das gibt

MERKSATZ

**Wenn ihr heute nur eine Sache mitnehmt: arbeitet parallel. Das Jonglieren übernimmt die App.**

COPILOT-SPEZIFISCH

# Ich zeige euch das kurz

Der Talk von heute Abend wandert auf liebeck.io auf „erledigt“ – live, aus der Copilot App heraus.

```
> Markiere den tech&talk-Vortrag vom 10.10.2026 auf liebeck.io als erledigt.
```

| | |
|---|---|
| 1 | Session im liebeck.io-Repo starten |
| 2 | Prompt absetzen – die App arbeitet |
| 3 | Änderungen prüfen & commit & merge |

***Genau dafür ist die App mein Cockpit: kleine Änderungen, komplett ohne IDE.***

01

# Tokens & Harness

## Warum reden wir überhaupt über Tokens?

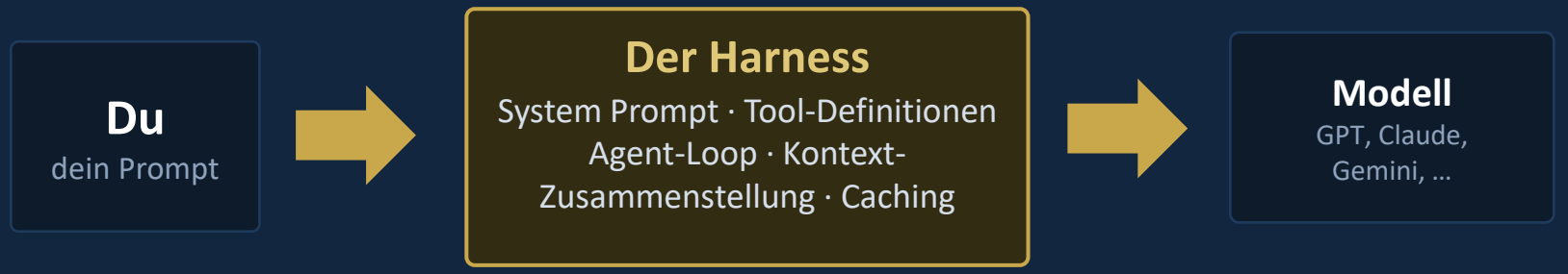
- Seit dem 01.06.2026 gibt es bei GitHub Copilot keine Flatrate mehr – abgerechnet wird nach Tokens
- Meine Prognose: Die anderen großen Anbieter ziehen nach – früh Bewusstsein schaffen lohnt sich
- Und wer Tokens versteht, versteht auch die Session-Limits bei Claude, Codex & Co.

KONZEPT PORTABEL

# Token-Arten & Token-based Billing

| Token-Art | Was ist das? | Warum relevant? |
|---|---|---|
| Input | Alles, was das Modell liest: Harness, Kontext, dein Prompt | 86 % eurer Credits kaufen Input – nicht Output |
| Output | Alles, was das Modell schreibt: Antworten, Code, Tool-Calls | Teurer pro Token, aber mengenmäßig klein |
| Cache Read | Bereits gecachter Kontext wird günstig wiedergelesen | Innerhalb einer Session zahlt der Cache zurück |
| Cache Write | Kontext wird für Wiederverwendung in den Cache geschrieben | Cache-TTL: 300 Sekunden |

- **86 % der Credits kaufen Input – über alle Modelle und Tasks (157 Läufe, 2 Produktions-Repos)**
- Jede neue Session zahlt die Baseline. Innerhalb einer Session zahlt der Cache zurück.



KONZEPT PORTABEL

# Das 20.000-Token-„OK“

*Prompt an Copilot CLI: „Antworte nur mit OK“*

| ~19.752 | 68 | 12,36 |
| --- | --- | --- |
| Input-Tokens | Tokens Konversation | AI Credits |

- Praktisch deterministisch: Median über 108 Läufe: 21.103 Tokens (19.787–23.328)
- Die eigentliche Konversation macht 0,3 % der Input-Tokens aus
- Gemessen: Copilot CLI 1.0.75, gpt-5.6-sol, session.shutdown-Events (events.jsonl)

MERKSATZ

**Gemessen in der CLI: ~20.000 Tokens bezahlt, bevor dein Prompt überhaupt zählt.**

KONZEPT PORTABEL

# Wohin gehen die Tokens?

Der Harness aus dem Einstieg – jetzt in Zahlen.

| | |
|---|---|
| ~70 % | **Tool-Definitionen (~14,1k Tokens)**<br>inkl. eingebautem GitHub MCP Server |
| ~30 % | **System Prompt (~7k Tokens)**<br>das Verhalten des Agenten |
| +3 % | **Repo-Kontext (~600 Tokens)**<br>Custom Instructions + Skill-Beschreibungen |

MERKSATZ

**Der Harness kostet, insbesondere zu Beginn einer neuen Session.**

◆THESE

## Advanced heißt nicht:
## bessere Prompts schreiben.

## **Advanced heißt: Workflows bauen.**

| **wiederverwendbar** | **persistent** | **kritisch geprüft** |
| --- | --- | --- |
| Skills | Memory Bank | Rubber Duck |

*… und mit dem Umstieg auf Token-based Billing bekommt jede dieser Entscheidungen ein Preisschild.*

02

# Custom Instructions & Custom Agents

Das Fundament in drei Minuten – und eine ehrliche Einordnung

KONZEPT PORTABEL

# Custom Instructions: das Fundament

- **.github/copilot-instructions.md – wird automatisch geladen: VS Code, Visual Studio, github.com, Copilot CLI, Copilot App und SDK**
- **Dasselbe Konzept bei den anderen: Claude Code liest CLAUDE.md, Codex liest AGENTS.md**
- Projekt-Konventionen einmal aufschreiben: Tech-Stack, Architektur, Do’s & Don’ts
- Wird mit jedem Request erneut mitgeschickt – ab dem zweiten Turn immerhin als günstiger Cache-Read
- **Deshalb: klein halten. Kurze, präzise Regeln statt Prosa**
- Grenze: immer geladen, für alle Tasks gleich – unpräzise bei speziellen Workflows
    - Genau diese Grenze lösen Skills (Block 03)

```
# GitHub Copilot Instructions for Gesellschaftsspieler-Gesucht

## Project Architecture
- This is an ASP.NET Core MVC application with Razor views (NOT Razor Pages, NOT Blazor)
- Target framework: .NET 9
- C# version: 13.0
- Follow Clean Architecture principles with layers: Web, Application, Domain, Infrastructure, Common

## Technology Stack
- Frontend: Razor views, jQuery, Bootstrap
- Backend: ASP.NET Core MVC with Entity Framework Core
- Database: SQL Server (assumed based on EF Core usage)
- Authentication: ASP.NET Core Identity
- Real-time: SignalR

## Code Style & Conventions
- Use German language for user-facing text (labels, validation messages, etc.)
- Use English for code (variable names, method names, comments)
- Controller names: Use `nameof(Controller).CleanController()` pattern in Razor views
- View names: Use `nameof(Controller.Action)` pattern in Razor views
- Follow existing naming conventions found in the codebase

## Razor View Guidelines
- Always use `asp-controller` and `asp-action` tag helpers for links and forms
- Use the `.CleanController()` extension method for controller names
- Maintain consistent Bootstrap grid layout: typically `col-lg-4` for labels, `col-lg-8` for inputs
- Use Font Awesome icons consistently (check existing usage patterns)
- Include validation spans: `<span asp-validation-for="PropertyName" class="text-danger"></span>`
- Use `@inject` for services like `SignInManager<User>` when needed

## ViewModels
- ViewModels should be in `Gesellschaftsspieler.Web.ViewModels` namespace
- Use suffix `VM` for ViewModel classes (e.g., `ProfileEditVM`, `GameVM`)
- Include display attributes for proper label rendering: `[Display(Name = "German Label")]`
- Use validation attributes appropriately
```

copilot-instructions.md · gesellschaftsspieler-gesucht.de

MERKSATZ

**Schreibt eine copilot-instructions.md. Alles Weitere baut darauf auf.**

KONZEPT PORTABEL

# Custom Instructions: Was sagt die Forschung?

## Kein Benchmark-Boost

Die Erfolgsquote auf realen Tasks steigt nicht messbar – die Kosten aber um ~20 %: mehr Tests, mehr Exploration, mehr Reasoning

## Aber: Anweisungen wirken

Instructions werden nachweislich befolgt (uv: 1,6× pro Task, wenn erwähnt – sonst <0,01×). Der Wert liegt in Konventionstreue, nicht in Benchmark-Punkten

## Selbst schreiben statt /init

Handgeschriebene Dateien schlagen LLM-generierte signifikant – auf generierte Context Files besser verzichten

## Nichts Redundantes

Nur rein, was nicht schon im README steht – Repo-Überblicke bringen nachweislich nichts

*Die Kehrseite bestätigt die Anekdoten: In kaum dokumentierten Repos helfen Context Files dagegen konsistent.*

MERKSATZ

**Die Studie bestätigt die Regeln von eben: klein halten, selbst schreiben, nichts doppeln.**

Gloaguen, Mündler, Müller, Raychev, Vechev: „Evaluating AGENTS.md“ · ETH Zürich / LogicStar.ai · arXiv 2602.11988 · Juni 2026 · SWE-bench + CtxBench, 4 Coding Agents

KONZEPT PORTABEL

# Custom Agents: das Konzept

- **Spezialisierte Agenten-Profile als .agent.md – im Repo oder auf User-Ebene**
- Eigene Persona, eigene Instruktionen, eingeschränkte Tool-Auswahl pro Agent
- Idee: „Review-Agent“, „Doku-Agent“, „Migrations-Agent“ statt ein Generalist
- Auswählbar im Agent-Dropdown – VS Code und Visual Studio 2026

```
---
name: readme-creator
description: Agent specializing in creating and improving README files
---

You are a documentation specialist focused on README files. Your scope is limited to README
files or other related documentation files only - do not modify or analyze code files.

Focus on the following instructions:
- Create and update README.md files with clear project descriptions
- Structure README sections logically: overview, installation, usage, contributing
- Write scannable content with proper headings and formatting
- Add appropriate badges, links, and navigation elements
- Use relative links (e.g., `docs/CONTRIBUTING.md`) instead of absolute URLs for files
within the repository
- Make links descriptive and add alt text to images
```

readme-creator.agent.md

KONZEPT PORTABEL

# Custom Agents: meine ehrliche Einordnung

**Beobachtung aus Projekten und Community:**

- Im Kontext von agentic coding hat sich das Feature in meinem Umfeld nicht durchgesetzt
- Der Mehrwert gegenüber Skills ist unklar:
  - beides kapselt Spezialwissen
  - Skills triggern automatisch – Custom Agents muss ich aktiv auswählen
- Für andere Agenten-Typen (außerhalb von agentic coding) kann das Konzept sinnvoll sein

MEINE EMPFEHLUNG

**Kennt das Konzept, aber investiert eure Energie in Skills. Sie lösen dasselbe Problem, triggern automatisch und sind portabel über Tools hinweg.**

03

# Agent Skills

Wissen einmal definieren – in jeder Session verfügbar

◆ SKILLS

PORTABEL

# Was sind Agent Skills?

- **Ein Ordner mit einer SKILL.md: Anleitung + optionale Skripte und Referenzen**
- Kapseln einen konkreten Workflow: „so machen wir bei uns Entity Framework Core-Migrationen“
- GitHub Copilot entdeckt Skills automatisch und führt sie aus, wenn dein Anliegen passt
- **Request-driven**: das LLM entscheidet anhand der Beschreibung, wann ein Skill ausgeführt wird
- Bei Claude Code zusätzlich explizit aufrufbar: per Slash-Befehl /{skill-name}, wenn du genau weißt, was du willst
- Zum Vergleich: Hooks sind event-driven (Datei-Save, Tool-Call) – heute nicht unser Thema

MERKSATZ

**Erstellt einen Skill, wenn ihr wiederholt die gleichen Schritte hintereinander ausführen wollt.**
**Ein Skill ist dokumentiertes Team-Wissen, das der Agent selbstständig findet und anwendet.**

PORTABEL

# Anatomie einer SKILL.md

```
ef-core-migrations/
├─ SKILL.md
├─ scripts/
└─ references/


---
name: ef-core-migrations
description: Erstellt und prüft
  EF-Core-Migrationen nach unseren
  Projekt-Konventionen
---
# Ablauf
1. dotnet ef migrations add …
2. Generierte Migration reviewen
3. …
```

- **Frontmatter entscheidet über den Trigger**
- name + description sind das Einzige, was im Leerlauf im Kontext liegt – die Beschreibung ist euer wichtigster Satz
- Der Body ist die eigentliche Anleitung: Schritte, Konventionen, Stolperfallen
- scripts/ für ausführbare Helfer, references/ für längere Doku

PORTABEL

# Wie Skills geladen werden – und was sie kosten

| | | |
|---|---|---|
| **1 · Leerlauf** | Nur Name + Beschreibung jedes Skills als frontmatter liegen im Kontext | Teil der ~600 Tokens Repo-Kontext (+3 %) |
| **2 · Trigger** | Dein Anliegen passt – der Agent liest die SKILL.md per Tool-Call | Kosten entstehen erst bei Nutzung |
| **3 · Ausführung** | Skripte & Referenzen werden nur bei Bedarf nachgeladen | Progressive Disclosure: nie alles auf einmal |

MERKSATZ

**Skills sind nicht preloaded: Im Leerlauf liegt nur die Beschreibung im Kontext. Der Skill kostet erst richtig, wenn er gebraucht wird.**

PORTABEL

# Beispiel 1: smartwarten-test-runner

1. Alle laufenden Docker Container beenden
2. Frischen Datenbank-Dump aus dem Azure Blob Storage herunterladen
3. Container neu bauen, starten – und auf den SQL-Server-Container mit dem Dump warten
4. Die komplette Playwright-E2E-Suite ausführen
5. Testergebnisse zusammenfassen

**TRIGGER-PHRASEN**

„rebuild and test“ · „run all tests“
„restart containers“ · „fresh database“

*Keine Polling-Skripte, keine Wait-Loops, kein Babysitting. Ein Prompt statt fünf manueller Schritte.*

```
---
name: smartwarten-test-runner
description: Use when the user asks to rebuild containers, run tests, restart Docker, or re-run the test suite. Triggers on
phrases like "rebuild and test", "run all tests", "restart containers", "fresh database".
---

# Docker Test Runner

Rebuilds SmartWarten Docker containers and runs the Playwright E2E test suite in one uninterrupted flow. No polling scripts,
no custom wait loops, no manual babysitting.
```

SKILL.md · SmartWarten-Repo

PORTABEL

# Beispiel 2: translations

1. Neue Textdatei im Format YYYYMMDD_feature-name.txt anlegen
2. Translation Keys mit deutschen Default-Werten eintragen: TranslationKey;GermanDefaultText
3. Python-Skript aufrufen – generiert die SQL-Statements für Insert und Update
4. Deutschen Text in die 8 weiteren Anzeigesprachen übersetzen und die Update-Statements aktualisieren

**TRIGGER LAUT DESCRIPTION**

UI-Labels · Translation Keys · Localization Strings

*Die Pointe: Das LLM übersetzt selbst. Der Skill gibt nur Format, Reihenfolge und Ablageort vor.*

**SO SIEHT DIE DATEI AUS**

```
0;TabletDashboard-PromptedTickets-Caption;Aufgerufene Tickets
1;QueueManagerDashboard-UnsuppressTicket-Title;Blockierung aufheben
```

```
---
name: translations
description: 'Use this skill when adding new UI labels, translation keys, localization strings, or when the user mentions translations,
localization, or multilingual text. This skill defines the correct workflow for creating translation entries across all supported languages.'
---

# Translation Workflow

## Overview

This skill defines the correct workflow for adding new translations to the SmartWarten application. Translations are managed via
semicolon-separated `.txt` files in the `/scripts` directory, which are processed by a Python script to generate SQL migration files.
```

SKILL.md · SmartWarten-Repo

PORTABEL

# Beispiel 3: ef-core-migrations

- Der Agent ändert nur das Model – Migrations-Dateien und Snapshot sind tabu, die generiert die EF Core CLI
- Am Ende: Migrationsnamen vorschlagen, Entwickler bestätigt, erst dann läuft das Kommando
- Sonderfall bei mir: DB-Zugangsdaten im Azure Key Vault – das Kommando braucht diesen Kontext

**INZWISCHEN OBSOLET**

Copilot ist besser geworden: Das Harness erstellt sich heute selbst eine IDesignTimeDbContextFactory, generiert die Migration und löscht die Factory wieder. Davor war der Skill mehrere Monate täglich im Einsatz.

***Skills automatisieren (1+2), begrenzen (3) – und dürfen in Rente gehen, wenn das Tool nachzieht.***

```
---
name: ef-core-migrations
description: 'Use this skill when making database model changes, adding or modifying entity
classes, updating DbContext configuration, changing Fluent API mappings, or when the user mentions
 migrations, database schema changes, or EF Core model updates. This skill prevents the agent from
 generating or editing migration files and snapshot files directly.'
---

# EF Core Migration Workflow

## Overview

This skill defines the correct workflow for making Entity Framework Core model changes in this
project. AI agents must never create or edit migration files or the model snapshot directly —
these files are generated by the EF Core CLI tooling.

## What You MUST Do

When the task involves database model changes:

1. **Make model changes only** — modify entity classes, DbContext configuration, and Fluent API
mappings as needed
2. **Do NOT create migration files** — never generate files in the
`src/SmartWarten.Domain/Migrations/` folder
3. **Do NOT edit the snapshot file** — never modify `DataContextModelSnapshot.cs`
4. **After all model changes are complete** — suggest a migration name and ask the developer to
confirm before running the migration command

## What You MUST NOT Do

- ❌ Create any `.cs` file inside `src/SmartWarten.Domain/Migrations/`
- ❌ Edit `DataContextModelSnapshot.cs`
- ❌ Edit any existing migration file (e.g., files starting with a timestamp like `20260215_`)
- ❌ Attempt to manually update the model snapshot to match your changes
- ❌ Run the migration command without asking the developer to confirm the migration name first
```

SKILL.md · SmartWarten-Repo

PORTABEL

# Discovery & Portabilität: einmal schreiben, überall nutzen

- **GitHub Copilot entdeckt Skills aus .github/skills/ – und auch aus .claude/skills/ und .agents/skills/**
- Derselbe SKILL.md funktioniert in GitHub Copilot UND in Claude Code – unverändert
- Format folgt der offenen Spezifikation: agentskills.io
- Visual Studio 2026: eigenes Skills-Panel im Chat + Built-in Skills der .NET- und Azure-Teams
- VS Code, Copilot CLI, Copilot App: gleiche Skills, gleiche Dateien

MERKSATZ

**Skills sind ein portables Format. Der Skill gehört euch, nicht dem Tool.**

PORTABEL

# Agent Plugins: vom Standard zum Feature

```
my-plugin/
├─ plugin.json
├─ skills/
├─ mcp.json
└─ com.github.copilot/
```

- **Offener Standard (agent-plugins.org): Spec 1.0 veröffentlicht im August 2026 – getragen von AWS, Anysphere, Google, Microsoft, OpenAI und Vercel**
- Bündelt Agent Skills und MCP-Server in EIN installierbares Paket
- **Seit August 2026 generally available: VS Code, Copilot CLI, SDK und Copilot App – auf allen Plänen**
- Marketplace ab Werk: github/awesome-copilot – Plugins direkt aus VS Code, CLI und App installieren
- Copilot-Spezifisches (Custom Agents, Hooks, Commands) liegt in com.github.copilot/ – andere Clients ignorieren es einfach

MERKSATZ

**Ein Paket für alle Tools. Wer heute Skills schreibt, schreibt schon im Format von morgen.**

04

# Memory Bank

Gedächtnis über Sessions hinweg – in Git statt in der Cloud

◆ MEMORY BANK

# Das Problem: Jede Session beginnt bei null

- Du erklärst deine Architektur … schon wieder
- Du erklärst deine Business-Logik … schon wieder
- Wissen aus einer Session überträgt sich nicht in die nächste
- Kontext, der 20 Minuten Aufbau gekostet hat, ist mit einem Klick weg
- *Symptom: alte Chat-Threads werden recycled, weil sie sich „aufgewärmt“ anfühlen*

Problem

**Aus Kapitel 1 wissen wir: Jede neue Session zahlt außerdem die volle Token-Baseline, aber lange Sessions sind auch teuer.**

COPILOT-SPEZIFISCH

# Kurz eingeordnet: das offizielle „Copilot Memory“

- Cloud-hosted, repo-scoped, lernt automatisch
- Erzeugt nur von Cloud Agent, Code Review und CLI – nicht vom IDE-Chat
- Memories verfallen nach 28 Tagen · ansehen & löschen ja, erstellen & editieren nein
- **Mein Test: 4 Memories nach mehreren PRs – technisch korrekt, aber zu granular, um zu helfen**

**Erhofft:** „Projekt nutzt deutsche Datumsformate“, „xUnit mit FluentAssertions“

**Gelernt:** wie EIN Feld in EINEM ViewModel geparst wird

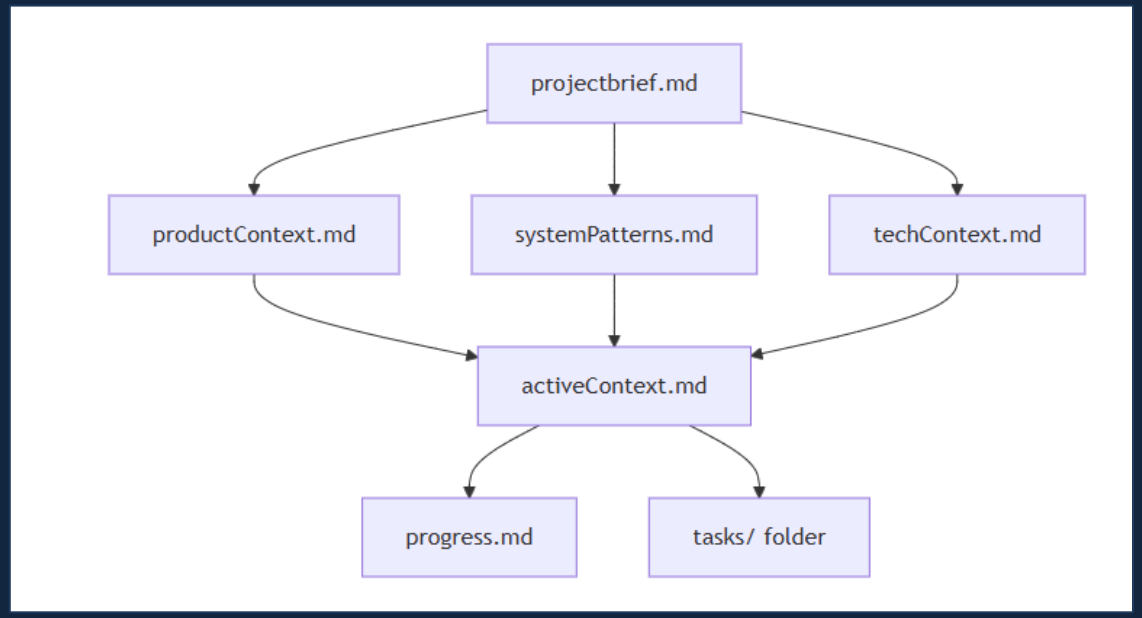
◆ MEMORY BANK

PORTABEL

# Das Memory-Bank-Pattern: Markdown statt Black Box

Ein Ordner Markdown-Dateien im Repo = Langzeitgedächtnis deines Agenten.
Lesen vor jeder Aufgabe, aktualisieren nach jeder Aufgabe.

| | |
|---|---|
| `projectbrief.md` | Ziele, Scope, Zielgruppe |
| `productContext.md` | Warum gibt es das Projekt? |
| `systemPatterns.md` | Architektur & Design Patterns |
| `techContext.md` | Tech-Stack & Constraints |
| `activeContext.md` | Aktueller Fokus, offene Punkte |
| `progress.md` | Erledigtes & nächste Schritte |




Struktur des Memory-Bank-Templates (awesome-copilot)

*Teamarbeit: activeContext.md & progress.md brauchen etwas Adaption – sonst drohen Merge-Konflikte.*

MERKSATZ

**Einstiegspunkt ist die copilot-instructions.md: „Lies /memory-bank/ vor jeder Aufgabe. Aktualisiere die memory Bank nach jeder Aufgabe.“**

◆ MEMORY BANK

# Copilot Memory vs. Memory Bank

| | Copilot Memory | Memory Bank |
|---|---|---|
| Speicherort | GitHub Cloud | Git-Repository |
| Erstellt von | KI automatisch | **Entwickler + KI gemeinsam** |
| Editierbar | nur ansehen & löschen | **volle Kontrolle** |
| Verfall | 28 Tage | **nie** |
| Transparenz | Black Box | **Markdown-Dateien** |
| Tool-Bindung | nur Copilot-Agenten | **tool-agnostisch: Copilot, Claude Code, …** |

Template: github/awesome-copilot → instructions/memory-bank.instructions.md

◆MEMORY BANK

PORTABEL

# In 10 Minuten startklar

| | | |
|---|---|---|
| 1 | **Template kopieren** | memory-bank.instructions.md aus github/awesome-copilot in eure .github/copilot-instructions.md |
| 2 | **Ordner anlegen** | /memory-bank/ im Repo erstellen |
| 3 | **Bootstrap-Prompt** | „Analysiere dieses Projekt und befülle den Memory Bank“ – im Agent Mode |

**DIE BERECHTIGTE FRAGE: „Input ist teuer – und ich soll pro Task sechs Dateien einlesen lassen?“**

**Die Alternative ist nicht „kein Kontext“, sondern: Kontext jedes Mal von Hand neu erklären – gleiche Tokens, plus eure Zeit.**

05

# Rubber Duck & Slash Commands

Die Ente redet zurück – und ihr automatisierter Nachfolger HydraFusion

COPILOT-SPEZIFISCH

# Rubber Duck: Die Ente redet zurück

- **Eingebauter Kritiker-Agent in GitHub Copilot – prüft Pläne, Designs, Code und Tests**
- **Der Clou: Der Kritiker läuft bewusst auf einem ANDEREN Modell als deine Session**
- Claude-Session → GPT-Kritiker (und umgekehrt): zwei unabhängige Perspektiven
- Anderes Modell = andere blinde Flecken. Genau das ist der Wert

COPILOT-SPEZIFISCH

# Wann die Ente sich meldet

## Automatisch, an Hebelpunkten:

- Nach dem Planen einer nicht-trivialen Änderung – vor der Implementierung
- Mitten in komplexer Arbeit
- Nach dem Schreiben von Tests
- Reaktiv: bei wiederholten Fehlschlägen
- *Triviale, gut verstandene Änderungen überspringt sie*

## Feedback, kategorisiert:

| | |
|---|---|
| **Blocking** | muss vor dem Weitermachen gelöst werden |
| **Non-Blocking** | sollte adressiert werden, blockiert nicht |
| **Suggestions** | Verbesserungsideen |

MERKSATZ

**Kein Gemecker über Stil, Formatierung oder Naming – nur Probleme, die den Erfolg der Aufgabe gefährden.**

COPILOT-SPEZIFISCH

# In der Praxis: /rubber-duck

```
> /rubber-duck Review unseren Plan für die 2FA-Implementierung.
  Finde blinde Flecken, Edge Cases und Risiken.
```

- Verfügbar in Copilot CLI und in der Copilot App – manuell per Slash Command oder Prompt
- Voraussetzung: Haupt-Session läuft auf einem Claude- oder GPT-Modell
- **Kosten-Ehrlichkeit: ein zusätzlicher Reasoning-Pass auf einem zweiten Modell – das kostet Latenz und Usage**
- Die Wette: Fehler im Plan zu finden ist billiger als drei fehlgeschlagene Implementierungs-Anläufe

COPILOT-SPEZIFISCH

# Frisch seit Donnerstag: HydraFusion

**Nachfolger der automatischen Modellwahl:** Auto wählte EIN Modell pro Request – HydraFusion wählt den Workflow, bei Bedarf mit mehreren Modellen

### Single Model

Ein Modell reicht – der Normalfall bleibt der Normalfall

### Cascade

Günstiges Modell entwirft, ein Quality Gate entscheidet über Eskalation

### Critique

Zweite Modell-Familie reviewt read-only – das Rubber-Duck-Prinzip, automatisiert

- GitHub-Angabe (nicht meine Messung): TerminalBench 2.1 +4,9 Punkte vs. Claude Opus 5, bei ~67 % geringeren Kosten
- Abrechnung: Token der tatsächlich genutzten Modelle zum Standardpreis – keine Orchestrierungs-Gebühr
- Heute nur Copilot CLI (/experimental) · App und VS Code laut GitHub als Fast Follow
- Claude kennt so etwas nicht: Dort entscheidest du dich vorab für ein Modell

MERKSATZ

**Auto wählte das Modell. HydraFusion wählt den Workflow.**

*Research Preview seit 04.09.2026 · Stand dieser Folie: 06.09.2026*

◆ BONUS

COPILOT-SPEZIFISCH

# Slash Commands in der Copilot App

### `/plan`

Aufgabe zerlegen, Ansatz durchdenken – schaltet in den Plan-Modus

### `/spar`

Copilot spielt zerpflückt deinen Ansatz kritisch (mit aktuellen Modell)

### `/autopilot`

Ziel übergeben, Copilot arbeitet die Schritte selbstständig ab

### `/rubber-duck`

Copilot reviewt unabhängig (mit zweiten Modell)

### `/create-canvas`

Interaktive Oberfläche direkt aus dem Chat erzeugen

### `/orchestrate`

Arbeit über mehrere Sessions und Repos koordinieren

Quelle: github.blog · „A guide to slash commands in the GitHub Copilot app“ (08/2026)

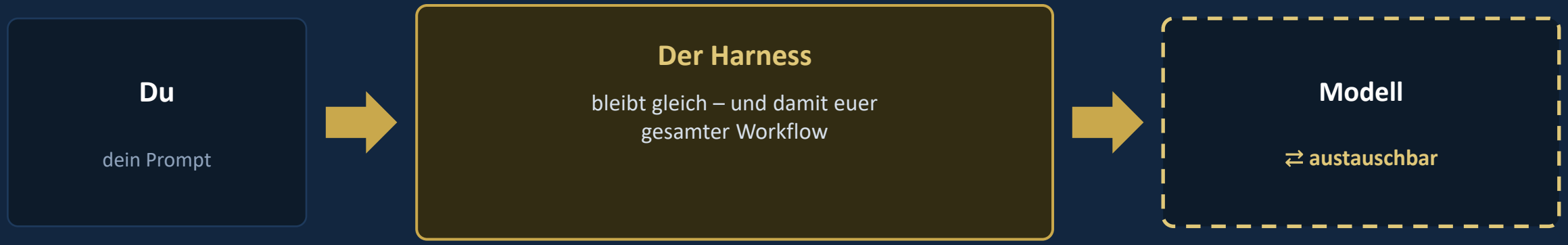
06

# Bring Your Own Model

Fünf Wege zu deinem Modell – und was davon wirklich praktikabel ist

KONZEPT PORTABEL

# Warum überhaupt ein eigenes Modell?




***Der Baustein rechts ist tauschbar. Der in der Mitte bleibt – ihr lernt nichts Neues, ihr tauscht nur das Modell.***

### Abrechenbarkeit

Eigener Vertrag, eigene Kostenkontrolle – Abrechnung direkt beim Anbieter

### Vorhandene GPU nutzen

Lokale Modelle ohne Cloud – wenn der Speicher mitspielt

### Compliance & Standort

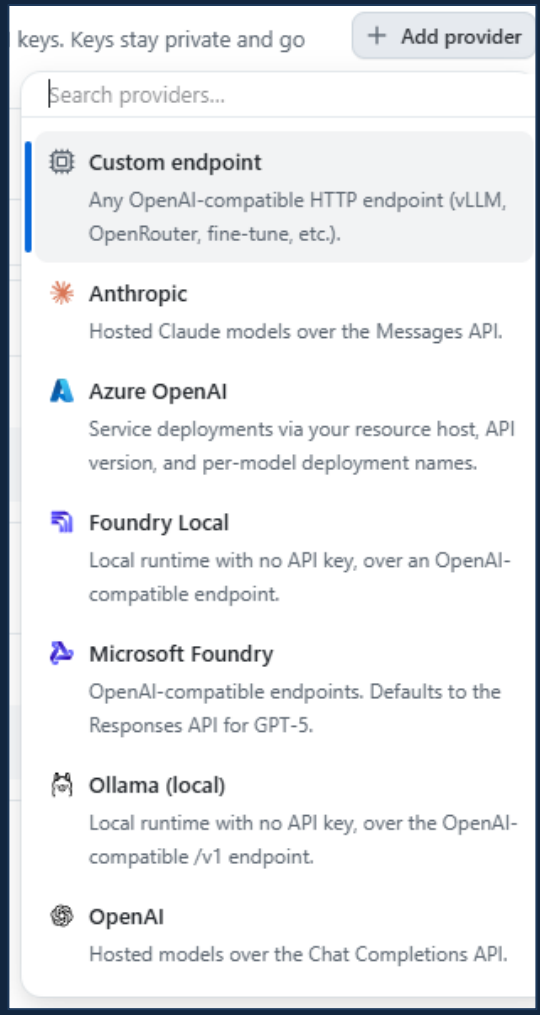
Nicht überall erlaubt: manche Teams dürfen nur on-premise oder in Europa

*Und gleich sehen wir: Es gibt fünf Wege dorthin.*

KONZEPT PORTABEL

# Fünf Wege zu deinem Modell

| | |
|---|---|
| **1 · Copilot-Modelle** | Standard: GitHub wählt und hostet – Abrechnung über Copilot-Credits |
| **2 · BYOK direkt** | Eigener API-Key (Anthropic, OpenAI, …) – Abrechnung beim Provider, zählt nicht gegen die Copilot-Quota |
| **3 · OpenRouter** | Ein Key, hunderte Modelle – auch große Open-Weight-Modelle, remote gehostet |
| **4 · Lokal (Ollama)** | Volle Kontrolle, Daten bleiben lokal – die Hardware entscheidet |
| **5 · Microsoft Foundry** | Eigenes Deployment im eigenen Tenant – Region wählbar, Abrechnung als Azure-Verbrauch |

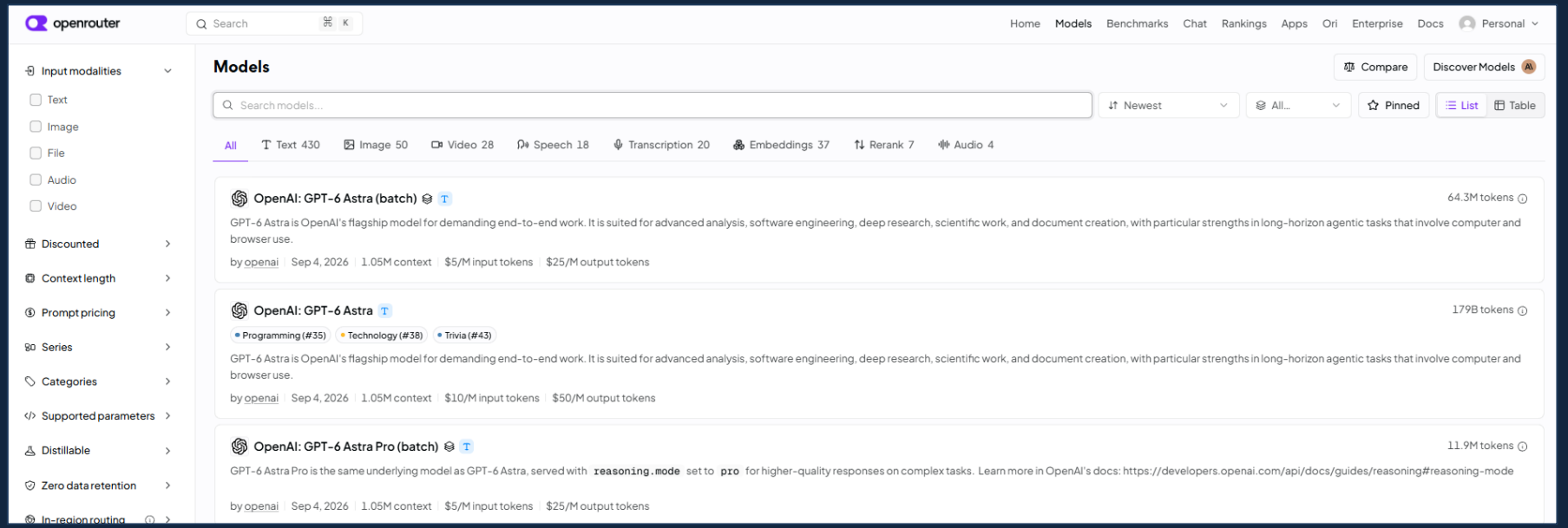



Add Provider in der Copilot App

PORTABEL

# OpenRouter: hunderte Modelle, ein Key

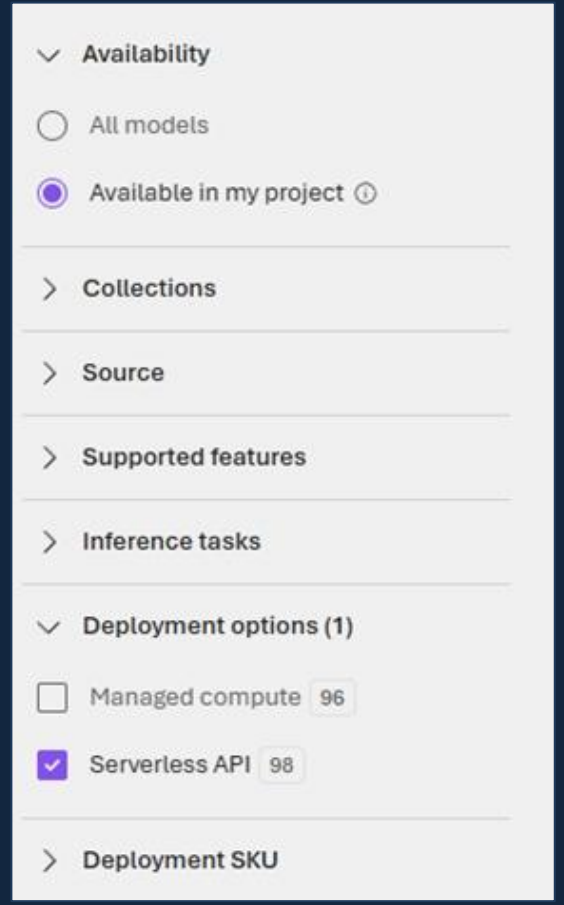
- **Aggregator, kein Modell-Anbieter: routet Requests an die dahinterliegenden Provider**
- Prepaid-Guthaben · Token-Preise 1:1 zum Listenpreis · leeres Guthaben stoppt die Calls
- Der eigentliche Gewinn: große Open-Weight-Modelle (DeepSeek, Qwen, Kimi) ohne eigene GPU
- **In der Copilot App: als Custom Endpoint (OpenAI-kompatibel) hinterlegen – selbst getestet mit qwen3-coder-next, stabiles Tool-Calling**



KONZEPT PORTABEL

# Weg 5 ausprobiert: eigenes Modell in Microsoft Foundry

- **DeepSeek-V4-Pro als Serverless-Deployment im eigenen Tenant, Endpoint plus Key in der App**
- Wichtigster Filter im Katalog: Serverless API (pay per token). Managed Compute mietet GPUs pro Stunde
- Qwen gäbe es hier nur als GPU-Miete: Der Speicher bleibt die Grenze, auch wenn er Microsoft gehört
- Enterprise-Winkel: Daten bleiben in Tenant und Region, Abrechnung als eigener Azure-Verbrauch
- **Getestet: echtes Feature auf dem eigenen Repo, stabiles Tool-Calling, gute Rückfragen**
- Stolpersteine: Project-Endpoint ist nicht der Inferenz-Endpoint (/openai/v1/), Wire API auf completions




Foundry-Katalog: Serverless API

# Die ehrliche Realität

## Lokal auf meiner Hardware (RTX 3050, 8 GB VRAM):

- Mechanik demonstrierbar: Ollama-Modell erscheint im Picker, antwortet – es funktioniert
- Produktiv agentisch arbeiten: nein – quantisierte 8B-Modelle scheitern an Tool-Calls
- Stolperfalle Windows: Kontextfenster über den UI-Slider setzen – sonst stille Trunkierung mitten im Wort

## Nachteile für den Firmeneinsatz:

- OpenRouter-Default-Falle: Free-Endpoint-Training überprüfen, zero data retention prüfen
- Die Garantie hängt am Endpoint, nicht am Modell – selbst erlebt: Qwen bestellt, Novita lieferte
- Business/Enterprise-Seats: BYOK läuft dort über die Admin-Ebene der Organisation

MERKSATZ

**BYOM zeigt: Ihr seid nicht an ein Modell gebunden. Aber der pragmatische Weg führt heute über die Cloud, nicht über eure GPU.**

◆ FAZIT

# Drei Dinge zum Mitnehmen

**1**

**Skills sind das Herzstück.**

Team-Wissen einmal definieren, automatisch getriggert, portabel über Tools – und im Leerlauf fast gratis.

**2**

**Gedächtnis gehört in euer Repo.**

Der Memory Bank ist editierbar, versioniert, permanent – alles, was die Cloud-Black-Box nicht ist.

**3**

**Lasst euch kritisieren – vom anderen Modell.**

Rubber Duck findet blinde Flecken, bevor sie teuer werden. Der zweite Reasoning-Pass ist die Wette wert.

*Alles davon hat ein Preisschild: Token-based Billing belohnt genau diese Workflows. Und: arbeitet parallel – Worktrees machen aus einem Agenten ein Team.*

◆ WEITERFÜHREND

# Tiefer einsteigen

### Where Did All My Credits Go?

Mein kompletter Talk zu Token-Ökonomie in Coding Agents – Agentic Coding Summit #2 (August 2026)

**Folien & Aufzeichnung: liebeck.io**

### MCP Servers in Production: OAuth, Rate Limiting & Quotas

Interne Systeme an Agenten anbinden – Azure Community Day Köln (04.09.2026)

**Folien: liebeck.io**

### GitHub Copilot Memory: From Black Box to Memory Bank

Der ausführliche Memory-Talk – Agentic Coding Summit #1

**Folien: liebeck.io**

### GitHub Copilot Newsletter

Agentic Coding & Copilot im .NET-Umfeld – ehrlich, getestet, regelmäßig

**ghcp.liebeck.io**

# Danke!

Fragen? Jetzt – oder auf LinkedIn.

LinkedIn

Dr. Matthias Liebeck · liebeck.io

GitHub Copilot Newsletter