

Bring Your Own Model: GitHub Copilot with Microsoft Foundry, OpenRouter and Local LLMs

Dr. Matthias Liebeck

Azure Cloud Native Köln · 13. Oktober 2026

◆ EINSTIEG

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



◆ ÜBERBLICK

Der Flugplan für heute

- 1 Einstieg: Agentic Coding und Harness
- 2 Warum es verschiedene Modelle gibt, und wer sie auswählt
- 3 Mehrere Wege zum eigenen Modell, mit OpenRouter und lokal
- 4 Microsoft Foundry im Detail
- 5 Entscheidungshilfe und Fazit

Die Frage des Abends: Wer entscheidet, welches Modell hinter deinem Copilot rechnet?

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 Anthrops Coding-Agent – Terminal, IDE-Anbindung und Desktop

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

◆ ZUM EINSTIEG

👉 Wer nutzt agentic coding?

👉 Und womit?

GitHub Copilot

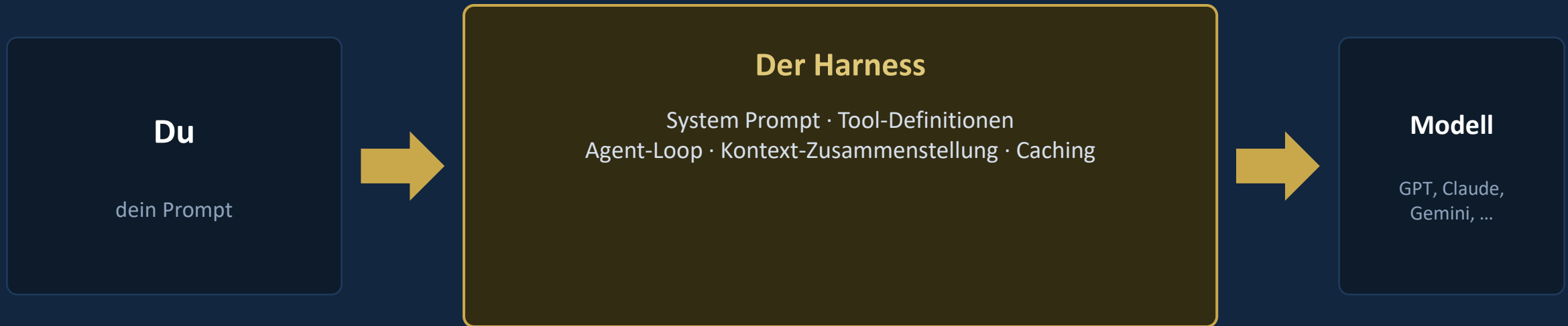
Claude Code

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 mit echten Zahlen.



Warum gibt es verschiedene Modelle?

Preis, Fähigkeit, und wer auswählt

◆ MODELLE

Modelle unterscheiden sich, auch im Preis

klein, schnell, günstig

groß, gründlich, teuer



- Faktor 40 zwischen dem günstigsten und dem teuersten Modell, bei Input wie bei Output
- Große Modelle sind meist stärker bei Planung und schwierigen Bugs, kleine schneller
- Dazu unterscheiden sich Kontextfenster, Anbieter und Datenort
- **Aber: Das günstigste Modell ist nicht automatisch der günstigste Coding-Agent. Jeder Fehlversuch kostet wieder Input**
- Input / Output in \$ pro 1 Mio. Tokens, GitHub Copilot, Stand 11.10.2026

Welche Aufgabe, welches Modell?

1

Routine: ein kleines Modell

Commit-Messages, Übersetzungen, Doku-Entwürfe

2

Planung und schwierige Bugs: ein großes Modell

Hier zählt sich gründliches Denken aus

3

Review: ein anderes Modell

Andere Modellfamilie, andere blinde Flecken. /rubber-duck macht genau das

4

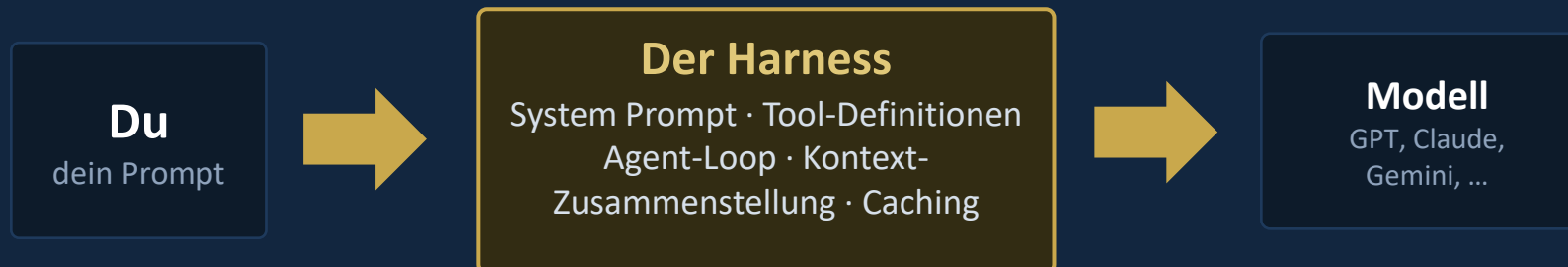
Bulk-Arbeit: der Stückpreis zählt

Viele kleine, gleichförmige Aufgaben

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.



Das 20.000-Token-„OK“

Prompt an Copilot CLI: „Antworte nur mit OK“

~19.752

Input-Tokens

68

Tokens Konversation

12,36

AI Credits

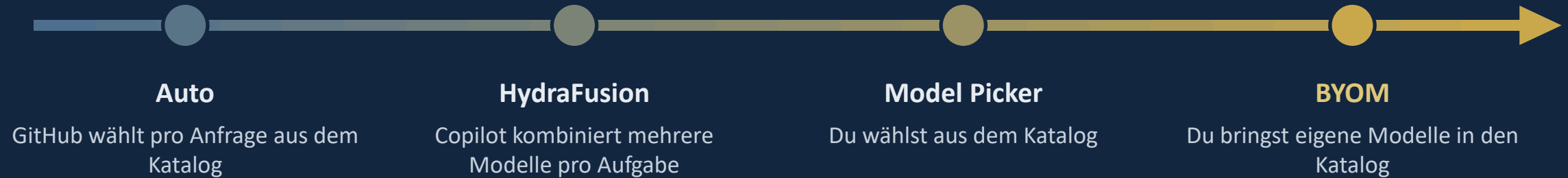
- Praktisch deterministisch: Median über 108 Läufe: 21.103 Tokens (19.787–23.328)
- Die eigentliche Konversation macht 0,3 % der Input-Tokens aus
- Gemessen: Copilot CLI 1.0.75, gpt-5.6-sol, session.shutdown-Events (events.jsonl)

◆ MODELLWAHL

Wer entscheidet, welches Modell rechnet?

kein Setup, GitHub entscheidet

eigener Vertrag, du entscheidest



- Alle vier laufen im selben Harness. Der Unterschied ist, wer auswählt und woher das Modell kommt
- BYOK, Bring Your Own Key: dein eigener Zugang zum Anbieter
- BYOM, Bring Your Own Model: das Modell, das du damit einbindest
- Ab hier sprechen wir von BYOM

Auto mit Stufen: GitHub wählt

Efficiency

Kosten niedrig, für schnelle, einfache Aufgaben

Balance

Kosten, Qualität und Latenz abgewogen, für den Alltag

Intelligence

Qualität zuerst, für komplexe Aufgaben

- **Alle drei Stufen nutzen dieselben Modelle. Auto bewertet jeden Prompt einzeln und wählt das passende Modell**
- Beispiel: Ein Docstring kann auch auf Intelligence ein kleines Modell bekommen
- Abgerechnet wird das gewählte Modell, egal welche Stufe. Zahlende Abos bekommen 10 % Rabatt
- Rollout in VS Code, Copilot CLI und App. Laut GitHub der erste Schritt zu mehr Einfluss auf die Modellwahl
- Bequem, aber GitHub entscheidet. Kein eigener Vertrag, keine eigene Region

HydraFusion: mehrere Modelle pro Aufgabe

Single

Ein gewähltes Modell löst die Aufgabe direkt

Cascade

Ein effizientes Modell legt einen Entwurf vor. Ein Quality Gate akzeptiert ihn oder eskaliert an ein stärkeres Modell

Critique

Ein Modell entwirft. Ein Kritiker aus einer anderen Modellfamilie prüft nur lesend, wie bei Rubber Duck. Dann wird einmal überarbeitet

- Die Wahl des Ablaufs ist ein Optimierungsproblem: Signale für Reasoning, Codegenerierung, Debugging und Tool-Nutzung bestimmen den günstigsten Ablauf, der die Qualitätsschwelle erreicht
- Auto wählt ein Modell pro Anfrage, HydraFusion einen Ablauf
- Abgerechnet werden die Tokens der genutzten Modelle zum normalen Preis, ohne Aufschlag
- Research Preview: Copilot CLI seit 04.09., App und VS Code seit 30.09.

HydraFusion: Zahlen und eigene Modelle

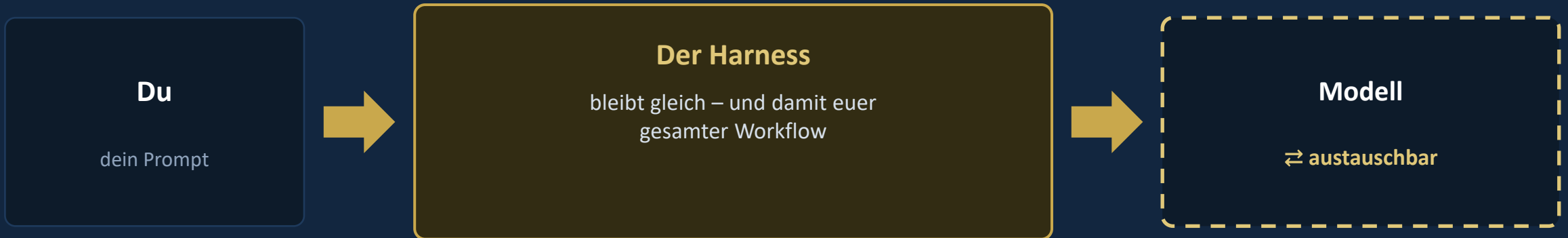
GitHub-Angabe, Basis Claude Opus 5	Kosten	Qualität
TerminalBench 2.1	−67 %	+4,9 Punkte
DeepSWE	−36 %	−1,5 Punkte
CheckpointBench (intern)	−65 %	−0,1 Punkte

- Deutlich günstiger, Qualität etwa auf Opus-Niveau
- Gemessen auf denselben Benchmarks, auf die HydraFusion abgestimmt wurde

Und eigene Modelle?

- HydraFusion wählt aus einem festen Pool von Modellen
- Lerna, ein Community-Plugin für die Copilot CLI, nennt sechs Modell-IDs: gpt-5.6-sol, -luna, -terra, claude-opus-5 und zwei MAI-Code-Varianten
- Lerna leitet die vier GPT- und Claude-Modelle auf eigene Foundry-Deployments um. Die MAI-Modelle bleiben bei Copilot
- Stand 11.10.: Mit BYOM geht HydraFusion nur teilweise

Warum überhaupt ein eigener Key bzw. 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 mehrere Wege dorthin.



Mehrere Wege zum eigenen Modell

Die gängigsten im Überblick

Die gängigsten Wege zu deinem Modell

1 · Copilot-Modelle

Standard: GitHub wählt und hostet – Abrechnung über Copilot-Credits

2 · Anbieter direkt

Eigener API-Key (Anthropic, OpenAI, ...) – Abrechnung beim Provider, zählt nicht gegen die Copilot-Quota

3 · OpenRouter

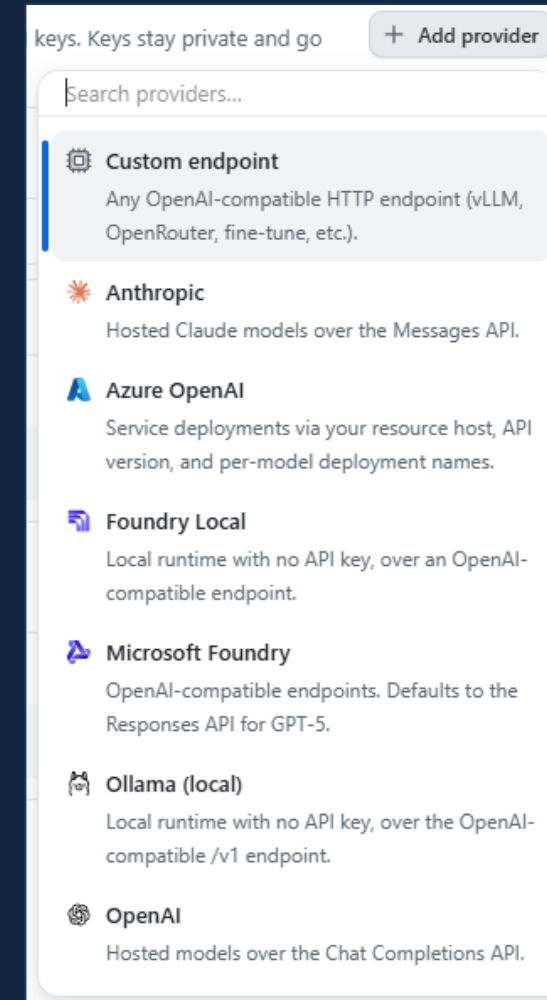
Ein Key, hunderte Modelle – auch große Open-Weight-Modelle, remote gehostet

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

Zwei Arten, eigene Modelle anzubinden

Persönliches BYOK

- Du trägst Key und Modell selbst im Client ein
- Der Key bleibt auf deinem Rechner
- Die Anfrage geht direkt zum Anbieter, ohne Copilot-API
- Geht auch ohne Copilot-Lizenz
- VS Code, JetBrains, Xcode, CLI, App, SDK
- Admins können es per Policy abschalten

Enterprise-BYOK

- Enterprise- oder Org-Owner hinterlegen Keys zentral
- Die Modelle erscheinen im Model Picker unter dem Firmennamen
- Die Anfrage läuft über die Copilot-API von GitHub
- Braucht Copilot Business oder Enterprise
- Copilot Chat, CLI und IDEs
- Public Preview

Wo eigene Modelle heute gehen

Oberfläche	Persönliches BYOK	Enterprise-BYOK	Selbst getestet
Copilot App	Ja, Preview	Ja	Ja
Copilot CLI	Ja	Ja	Ja
Copilot SDK	Ja	Ja	Ja
VS Code, JetBrains, Xcode	Ja	Ja	Nein
Visual Studio 18.10	Neuer Agent, Preview	Ja	Nein
Copilot Chat auf github.com	Nein	Ja	Nein

Quellen: GitHub-Doku, Visual-Studio-Release-Notes, Stand Oktober 2026

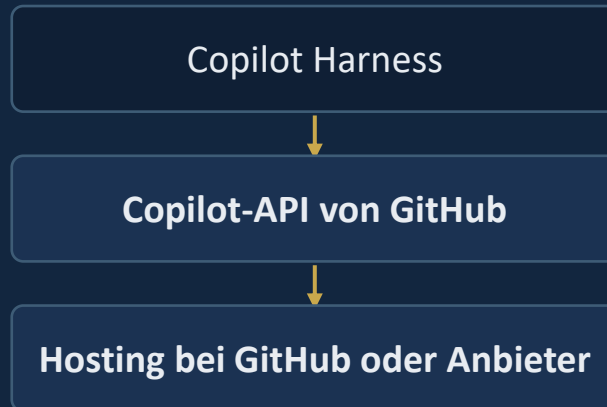
Was BYOM nicht abdeckt

- Inline-Completions laufen weiter über Copilots eigene Modelle
- Indexierung und Codesuche können weiter über Copilot laufen. Dein Modell bekommt die gefundenen Stellen als Kontext
- Enterprise-BYOK: Die Copilot-Lizenz bleibt, die Tokens zahlst du zusätzlich beim Anbieter
- Filter und Datenschutz hängen davon ab, ob die Anfrage über die Copilot-API läuft oder direkt zum Anbieter
- Generell gilt: vieles ist Public Preview und ändert sich wöchentlich

BYOM ersetzt das Modell, nicht das Produkt.

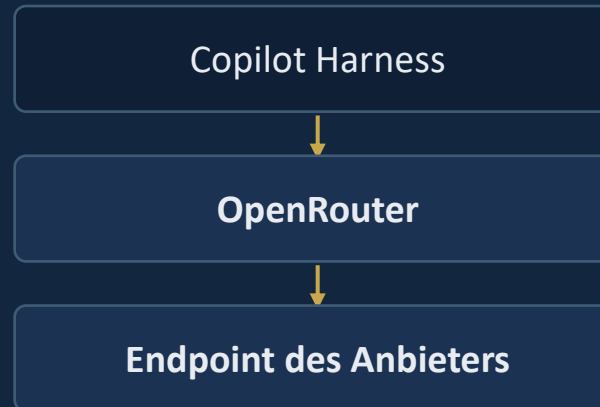
Wo fließen meine Prompts hin?

GitHub-Modell



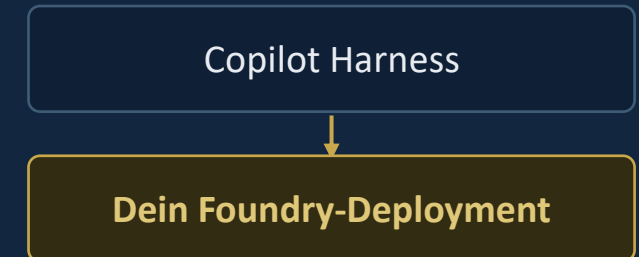
Sieht den Prompt: GitHub und ggf. der Modellanbieter

OpenRouter



Sieht den Prompt: OpenRouter und der Anbieter. Speicherung je nach Einstellung

Microsoft Foundry

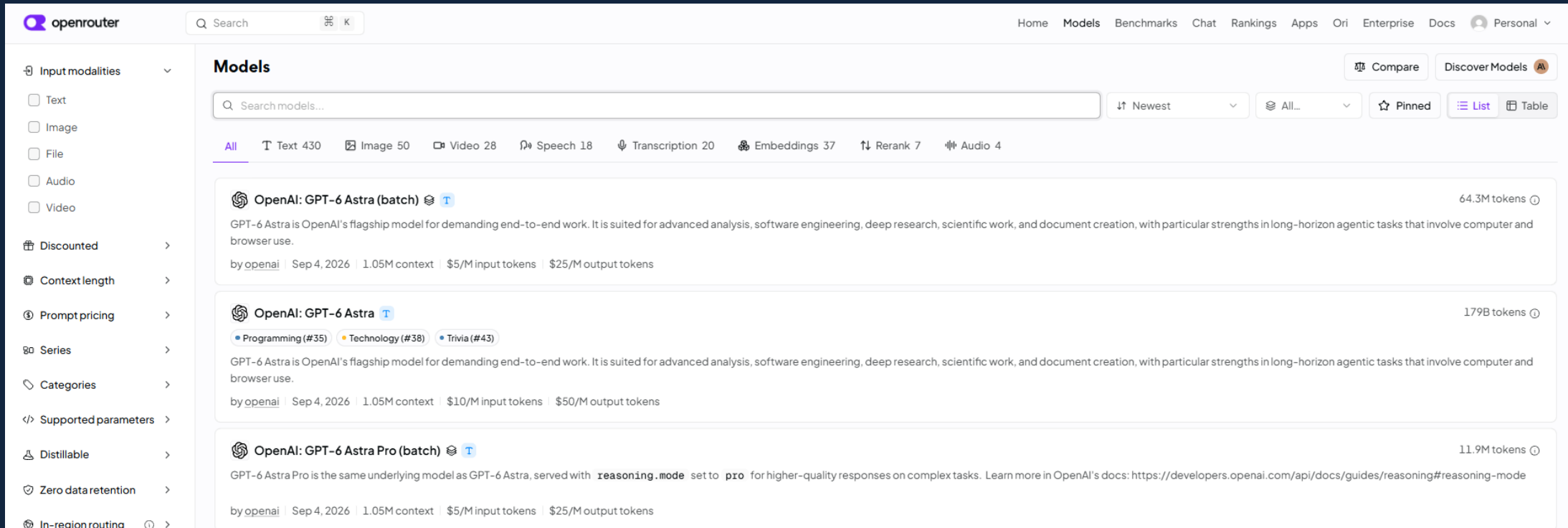


Sieht den Prompt: Microsoft, Ort je nach Deployment-Typ

- OpenRouter und Foundry hier mit persönlichem BYOK: direkt zum Anbieter. Enterprise-BYOK läuft zusätzlich über die Copilot-API
- Ressource ist nicht gleich Verarbeitungsort: Mein Luna-Deployment liegt in Sweden Central, als Global Standard kann weltweit verarbeitet werden
- Gezeigt ist nur der Inferenzpfad. Telemetrie, Indexierung und MCP-Tools haben eigene Wege

OpenRouter: hunderte Modelle, ein Key

- **Aggregator, kein Modell-Anbieter:** routet Requests an die dahinterliegenden Provider. Kosten pro Request auf der Activity-Seite
- Prepaid-Guthaben · Listenpreise plus 5,5 % Plattformgebühr beim Aufladen (Pay as you go) · 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: Tool Calling lief stabil**



The screenshot shows the OpenRouter website interface. On the left is a sidebar with navigation options like 'Input modalities', 'Discounted', 'Context length', 'Prompt pricing', 'Series', 'Categories', 'Supported parameters', 'Distillable', 'Zero data retention', and 'In-region routing'. The main content area is titled 'Models' and features a search bar, sorting options (Newest, All, Pinned, List, Table), and a list of models. The first model listed is 'OpenAI: GPT-6 Astra (batch)' with 64.3M tokens. The second is 'OpenAI: GPT-6 Astra' with 179B tokens. The third is 'OpenAI: GPT-6 Astra Pro (batch)' with 11.9M tokens. Each model entry includes a description, release date, context length, and pricing.

Ein Modell, viele Anbieter

Kimi K3, \$ pro 1 Mio. Tokens	Input	Output
Günstigster Input (Wafer)	0,33	14,89
Günstigster Output (Decart, mit Rabatt)	1,98	9,90
Listenpreis Moonshot und GitHub Copilot	3,00	15,00
Teuerster Anbieter (Fireworks Fast)	4,50	22,50

- Dasselbe Modell: Input 0,33 bis 4,50 \$, Output 9,90 bis 22,50 \$ je Endpoint
- Der günstigste Input ist nicht der günstigste Output. 86 % der Credits kaufen Input, deshalb zählt vor allem der Input-Preis
- Anbieter unterscheiden sich auch in Quantisierung, Geschwindigkeit, Kontextfenster und Zuverlässigkeit. Anbieter festpinnen
- OpenRouter-Modellseite, Stand 11.10.2026

OpenRouter: Wer verarbeitet meinen Code?

Risiken

- Ein zusätzlicher Dienst zwischen Copilot und Anbieter
- Dasselbe Modell bei verschiedenen Anbietern, mit verschiedenen Datenschutzregeln
- Routing und Fallbacks können den Anbieter wechseln. Selbst erlebt: Qwen bestellt, Novita lieferte
- Kein Training, keine Speicherung und EU-Verarbeitung sind drei verschiedene Anforderungen

Die Schalter dagegen

- Zero Data Retention: nur Endpoints, deren ZDR-Richtlinie keine dauerhafte Speicherung von Prompts und Antworten vorsieht
- Data Collection Deny: Anbieter ausschließen, die Daten sammeln oder damit trainieren
- Provider-Allowlist: nur freigegebene Anbieter, Fallbacks aus
- Region: Verarbeitung eingrenzen, wo Account und Anbieter das unterstützen

Aus Copilot heraus setzt du die Schalter in den OpenRouter-Einstellungen, nicht pro Request

Ein Modellname ist keine Datenschutzgarantie. Entscheidend ist der Endpoint, der antwortet.

Lokale Modelle

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

- Die Mechanik funktioniert: Das Ollama-Modell erscheint im Picker und antwortet
- Mein Test: Mit quantisierten 8B-Modellen war agentisches Coding nicht zuverlässig genug, vor allem beim Tool Calling
- Stolperfalle Windows: Kontextfenster über den UI-Slider setzen, sonst wird still mitten im Wort abgeschnitten

Neu seit 7. Oktober, nicht selbst getestet:

- Copilot CLI: /model findet laufende Ollama-Modelle, Wechsel ohne Neustart
- Voraussetzung: Das Modell kann Tool Calling und Streaming
- Lokales Modell heißt nicht, dass Copilot offline arbeitet. Telemetrie und andere Netzwerkzugriffe separat prüfen

MERKSATZ

Lokale Modelle funktionieren technisch. Für Coding-Agenten bleibt meine Hardware bislang die Grenze.



Weg 5: Microsoft Foundry

Weg 5 ausprobiert: eigenes Modell in Microsoft Foundry

- **DeepSeek-V4-Pro als Serverless-Deployment im eigenen Tenant, Endpoint plus Key in der App**
- Für meinen Test: Serverless API mit Pay-per-Token. Managed Compute heißt eigene GPU-Kapazität, abgerechnet nach Zeit
- Qwen3-Coder-Next gab es bei meinem Test nur über Managed Compute
- Enterprise-Winkel: Deployment und Abrechnung im eigenen Tenant. Wo verarbeitet wird, entscheidet der Deployment-Typ
- **Getestet: echtes Feature auf dem eigenen Repo, stabiles Tool-Calling, gute Rückfragen**

The screenshot shows a configuration page for a deployment in the Microsoft Foundry console. It features several expandable sections:

- Availability:** Contains two radio button options: "All models" (unselected) and "Available in my project" (selected, indicated by a purple dot).
- Collections:** A section with a right-pointing chevron.
- Source:** A section with a right-pointing chevron.
- Supported features:** A section with a right-pointing chevron.
- Inference tasks:** A section with a right-pointing chevron.
- Deployment options (1):** Contains two checkbox options: "Managed compute" (unselected, with a badge showing "96") and "Serverless API" (selected with a purple checkmark, with a badge showing "98").
- Deployment SKU:** A section with a right-pointing chevron.

Foundry-Katalog: Serverless API

Was kostet BYOM wirklich?

GPT-5.6 Luna, \$ pro 1 Mio.	Input / Output	Aufschlag	Kosten sehen
GitHub Copilot	0,20 / 1,20	keiner	GitHub Usage
OpenRouter	0,20 / 1,20	5,5 % beim Aufladen	Activity-Seite
Azure EU als Anbieter bei OpenRouter	0,22 / 1,32	Data Zone EU ca. +10 %	Activity-Seite
Microsoft Foundry	je Deployment-Typ, Preisrechner	Global, Data Zone, Regional	Azure Cost Management

- Listenpreise sind nur bedingt vergleichbar: Copilot-Kontingente, Tokenabrechnung und Guthabengebühren unterscheiden sich
- GitHub und OpenRouter-Modellseite, Stand 11.10.2026

Der Tokenpreis ist nicht der einzige Unterschied. Cache, Deployment-Typ und Gebühren zählen mit.

Modelle kommen und gehen

- 10.09.: GitHub kündigt MAI-Code-1-Flash ab
- 02.10.: Kimi K2.7 Code, Claude Opus 4.7, Gemini 3.5 und 3.6 Flash folgen
- Mit eigenem Deployment entscheidest du, wann du wechselst
- Auch Foundry nimmt Modelle raus, etwa DeepSeek-V3.1 aus dem Model Router

◆ WEG 5 · FOUNDRY

Wann lohnt der eigene Tenant?

1

Deployment-Kontrolle: Prompts gehen an dein eigenes Deployment. Wo verarbeitet wird, entscheidet der Deployment-Typ: Global, Data Zone oder Regional.

2

Version gezielt wählen: Abo-Modelle können sich serverseitig ändern. Das eigene Deployment bleibt innerhalb der Lifecycle-Fristen stabil. Wer misst und vergleicht, braucht genau das.

3

Kostenmodell: pay per token auf der Azure-Rechnung statt AI Credits: eigener Vertrag, eigene Kostenkontrolle.

4

Modellwahl: der Foundry-Katalog reicht über die Abo-Modelle hinaus.

BYOM ist kein automatisches Qualitäts-Upgrade, sondern mehr Kontrolle über Modellwahl, Betrieb und Kosten.

Foundry im Unternehmen

Azure-Governance

- Eigene Subscription, zentrale Abrechnung
- Deployment-Typ bewusst wählen: Global, Data Zone oder Regional
- Budgets und Alerts. Sie stoppen keine Requests
- Content-Filter pro Deployment prüfen
- Bei Partner-Modellen genau hinsehen, wer hostet

GitHub-Governance

- Enterprise-BYOK: Keys zentral hinterlegen
- Modelle pro Organisation freigeben
- Copilot Business oder Enterprise bleibt nötig
- Die Copilot-API bleibt im Inferenzpfad
- Public Preview, nicht selbst getestet



Fazit

Welcher Weg passt zu mir?

Kein Setup, kein zweiter Vertrag?



Weg 1: Copilot-Modelle, gern mit Auto

Bestehender Vertrag bei Anthropic oder OpenAI?
(und viel Geld für tokens vorhanden 😊)



Weg 2: Anbieter direkt

Viele Modelle ausprobieren?



Weg 3: OpenRouter

Viel GPU-Speicher, Daten dürfen nicht raus?



Weg 4: Lokal

Azure bereits im Einsatz, zentrale Governance oder
Anforderungen an die Datenresidenz?



Weg 5: Foundry, Deployment-Typ prüfen

Die ersten 30 Minuten mit Foundry

1 Budget mit Alert anlegen

Warnt bei Schwellenwerten, stoppt aber keine Requests

2 Modell serverless deployen

Modell mit Tool Calling, Deployment-Typ bewusst wählen: Global, Data Zone oder Regional

3 Provider in der Copilot App eintragen

Inferenz-Endpoint, nicht den Projekt-Endpoint

4 Eine kleine echte Aufgabe

Unkritisches Repo, danach Tool Calling, Tokenverbrauch und Kosten prüfen

Cost Management zeigt die Kosten mit einigen Stunden Verzögerung. Für den ersten Test reicht das.

Drei Sätze zum Mitnehmen

1 Copilot liefert den Harness.

Agenten-Loop, Tools und Workflow bleiben gleich, egal welches Modell rechnet.

2 BYOM gibt dir die Modellwahl.

Modelle unterscheiden sich um den Faktor 40 im Preis. Das günstigste Modell ist nicht automatisch der günstigste Agent.

3 Foundry bringt Deployment und Abrechnung in deinen Azure-Tenant.

Wo verarbeitet wird, entscheidet der Deployment-Typ.



Danke



Connect on LinkedIn

linkedin.com/in/matthias-liebeck



GitHub Copilot Newsletter

ghcp.liebeck.io