Nützliche Skills & MCPs für Entwickler

Agentic Coding-Meetup Bonn

Marius Shekow · 3. September 2026

Agenda

1. Nützliche Skills & MCPs für Entwickler
2. Nützliche Skills
3. Codebase-Intelligenz
4. Aktuelle Dependencies
5. Aktuelle Dokumentation
6. Effizientes Reviewen von Plänen oder Code
7. Coding Agent Abo vs. Coding Agent
8. Verarbeitung von Videos und Bildern
9. Kosten verstehen und überblicken

Nützliche Skills

Was ist ein Skill?

- Eine `SKILL.md`-Datei, die der Agent **nur bei Bedarf** nachlädt – passend zum aktuellen Prompt
- Ziel: gezielt Modell-Schwächen ausgleichen, ohne den Kontext dauerhaft zu belasten
- Zwei Praxisbeispiele auf der nächsten Folie: **Sprachstil** und **Requirements**

Praxisbeispiele

Weg mit den "Claudisms"

Zu blumige, unnötig komplizierte Formulierungen? Diese Skills trimmen den Schreibstil auf den Punkt:

- stop-slop
- SimpleEnglish

Sauber nachfragen

Manche Modelle/Agenten hinterfragen Anforderungen zu wenig – dieser Skill erzwingt es:

- Grilling Skill von Matt Pocock

Nur nötig, falls Modell/System-Prompt das nicht schon von sich aus tun

1000+ Skills warten auf skills.sh darauf, entdeckt zu werden

Braucht man Skills?

★ Starke Frontier-Modelle

- Können vieles davon **eh schon**
- Skills bringen hier oft wenig Zusatznutzen
- Nur dann nützlich, wenn **eigene Vorlieben** umgesetzt werden sollen

💵 Günstige China-Modelle

- Liegen 3–6 Monate hinter den Frontier-Modellen
- Skills können das Ergebnis **deutlich** verbessern
- Messbar, z. B. via agent-skills-eval


Codebase-Intelligenz

Wie durchsucht dein Agent den Code?

✗ `grep`

Stumpfe Textsuche, kein Verständnis von Struktur

− LSP

Kennt Symbole & Referenzen, aber keinen echten Graphen

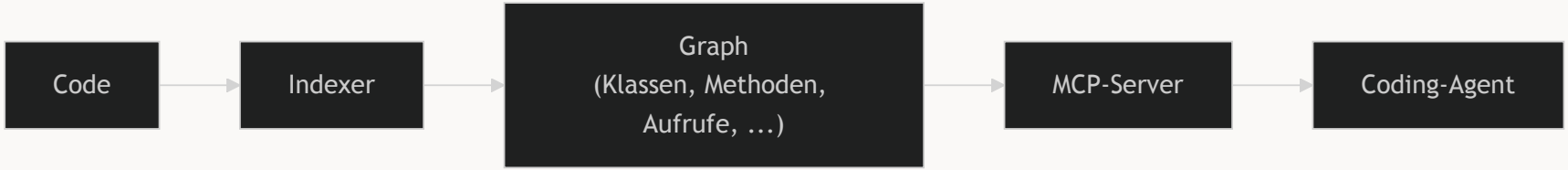
✓ Graph-Index

Versteht Beziehungen zwischen Code-Elementen

Nicht jeder Coding-Agent bringt von Haus aus einen guten Indexer mit

So funktioniert ein Indexer

- Läuft als **eigenes Tool** neben dem Coding-Agenten und hält den Graphen in Echtzeit aktuell
- Wird vom Agenten per **MCP** angesprochen
- Sogar **technologieübergreifend**: erkennt z. B. einen HTTP-Call eines Python-Clients auf einen Java-Spring-Boot-Server



- Findet dank **semantischer** Suche die Klasse `` `Authenticate` ``, obwohl du nach "Login" gefragt hast

Vorteile

- **Token-effizientere** Suche
- In Einzelfällen: bessere Treffer

Tools

- codebase-memory-mcp
- codegraph

Stolpersteine

- ⚠️ Der Agent muss manchmal "gebogen" werden
  - Beispiel: OpenCodes **Plan**-Agent nutzt lieber seinen eigenen *Explore*-Agenten statt den MCP-Indexer
  - Verhalten hängt stark vom eingesetzten LLM und dem Systemprompt ab
- 💡 Bonus-Tipp: auch *textuelles* Wissen (Markdown-Dateien) lässt sich so durchsuchbar machen, z. B. mit QMD


Aktuelle Dependencies

Problem

Software hängt an Third-Party-Dependencies (React, express, Python's `yaml`-Parser, …)

LLMs kennen nur ihren Trainingsstand → in generierten Manifesten wie `package.json` landen **fast immer veraltete Versionen**

Lösung

Ein MCP, der live die aktuellsten Versionen ermittelt:

- package-version-check-mcp


Aktuelle Dokumentation

Code ist nur so gut wie die Doku dahinter

Generierter Code muss die Dependency **korrekt** aufrufen – doch LLMs kennen entweder nicht die aktuelle Doku-Version, oder ihr "Wissen" darüber ist komprimiert und verlustbehaftet.

Warum nicht einfach `Web search`?

- € Teuer und langsam
- Doku-Formate variieren stark von Library zu Library
- Oft zu umfangreich fürs Context Window
- LLMs halluzinieren häufig die Quell-URLs

Lösungsansatz: vorindizierte Doku

Ein Tool (CLI/MCP), das für eine Query wie *"wie mache ich X in Dependency Y"* die passendsten Doku-Snippets als Markdown liefert.

Selbst bauen? Aufwändig – für jede Dependency einzeln recherchieren. Oder "von der Stange": 🌐 **Context7**

100.000+ Dependency-Dokus, die Context7 semantisch indiziert hat – Libraries **und** Plattformen (z. B. Vercel, Render.com)

Worauf achten?

- Kostenloser Account nötig – 1.000 Aufrufe/Monat inklusive
- Agent nutzt Context7 oft erst nach Nudge in `` `AGENTS.md` `` bzw. im Prompt
- Manche Anbieter haben **bessere eigene** Doku-MCPs:
  - Microsoft Learn MCP (Azure, Windows, Office, Power Platform, …)
  - aws AWS Docs MCP

Effizientes Reviewen von Plänen oder Code

Das Präzisionsproblem

Bei Tasks, die groß genug für eine **Plan-Phase** sind, will man den (kurzen) Plan reviewen, bevor viel Code entsteht.

Änderungswünsche müssen **präzise** sein – sonst ändert der Agent etwas Falsches:

> "In Klasse `FooBar` ist der Algorithmus in Zeile 342 nicht optimal, …"

> "In Phase 2 soll die Klasse `AuthenticateModule` und nicht `Authenticator` heißen, und bei Phase 3 kannst du Step 2+3 weglassen, …"

– typische, aber mühsame Freitext-Korrektur

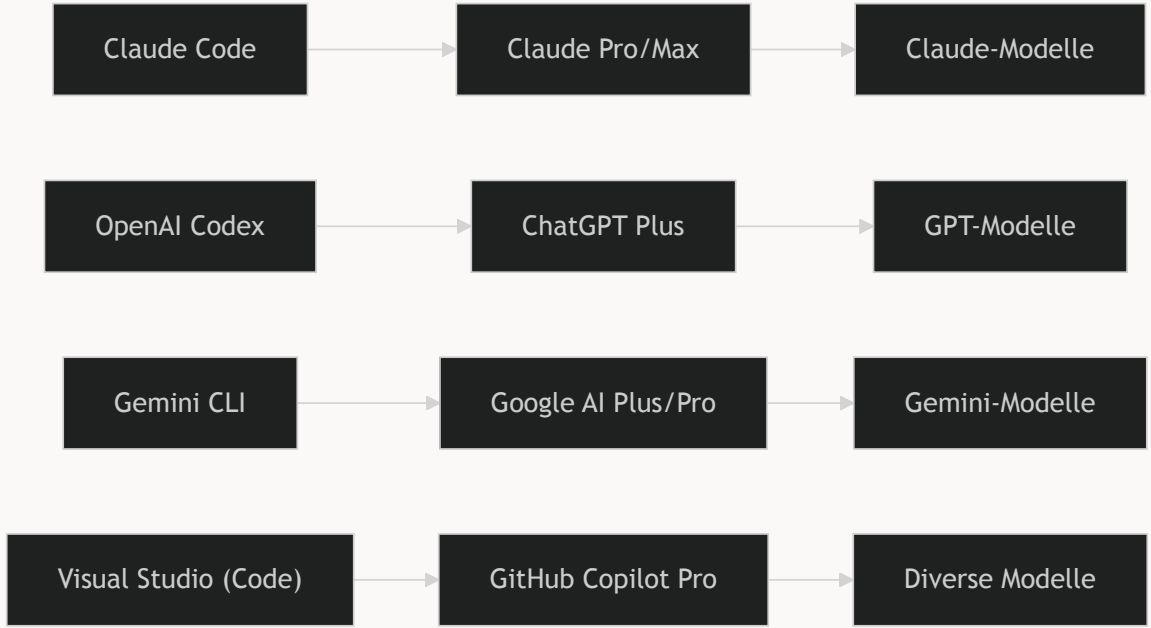
Lösung: Plannotator

- 🌐 Plannotator startet einen lokalen Web-Server mit einer PR-artigen Review-Oberfläche
- 📝 Man kommentiert / löscht gezielt einzelne Abschnitte als Annotation
- Zwei mögliche Ausgänge:
  - ✅ Keine (oder nur triviale) Änderungen → Plan freigegeben, Agent kann bauen
  - 🔄 Größere Änderungen → Plan wird neu erzeugt, Plannotator zeigt Vorher-Nachher-Diff zur erneuten Prüfung

⇄ Coding Agent Abo vs. Coding Agent

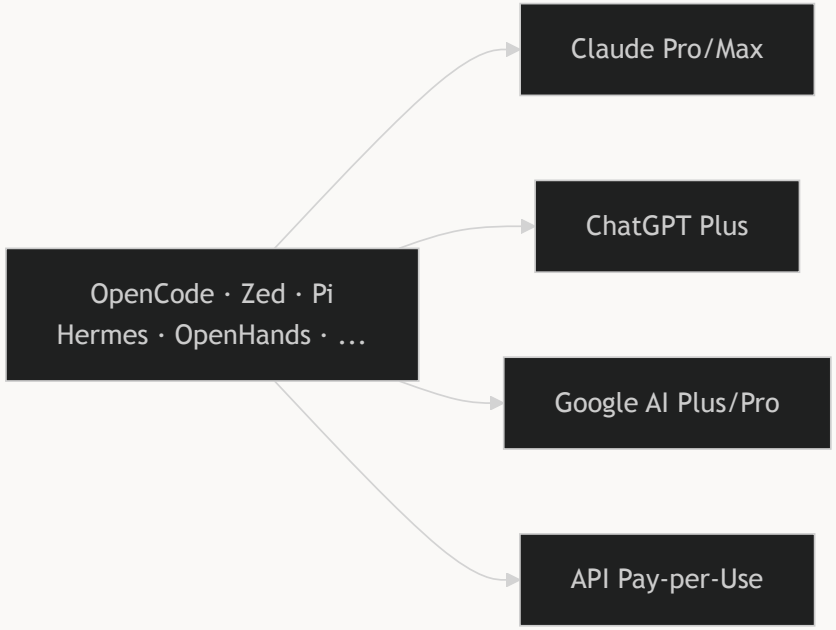
Die Versuchung der "offiziellen" Kopplung

Jedes Abo (Claude Pro, ChatGPT Plus, Google AI Plus/Pro, GitHub Copilot Pro) legt nahe: nutze den Coding-Agenten **desselben** Anbieters.



Es geht auch anders

- **Unabhängige Agenten** können für dein Projekt besser passen als der "offizielle" Agent
- Sie unterstützen i. d. R. dein bestehendes Abo trotzdem
- Ist dein Haupt-Abo-Kontingent aufgebraucht? Per Klick zu einem anderen Anbieter (oder einer Pay-per-Use-API) wechseln



Verarbeitung von Videos und Bildern

Use Cases

- 🐞 Bug-Triage anhand von Screenshots/Videos
- 🎨 Prüfen, ob ein Design-System eingehalten wurde
- 🎬 Screencasts in End-to-End-Tests verwandeln

Nicht jedes Modell "sieht"

Flagship-Modelle verarbeiten Bilder problemlos – günstige China-Modelle (GLM, DeepSeek, …) oft nicht

Lösungsansatz

Spezialisierte Vision-Modelle wandeln Bilder/Sequenzen in Text um – auch lokal & kostenlos:

- claude-video
- opencode-senses

Kosten verstehen und überblicken

Ey Mann, wo is’ mein Quota?

Claude-Code-Nutzer sind immer wieder überrascht, wie schnell ihr Quota aufgebraucht ist.

Abhilfe: die Statuszeile selbst anpassen, per Prompt:

```
/statusline <wie soll sie aussehen und welche Infos beinhalten>
```

So zeigt sie z. B. den verbrauchten Quota-Anteil und den nächsten Reset-Zeitpunkt an – inklusive Farben.

Tools zur Kostenanalyse

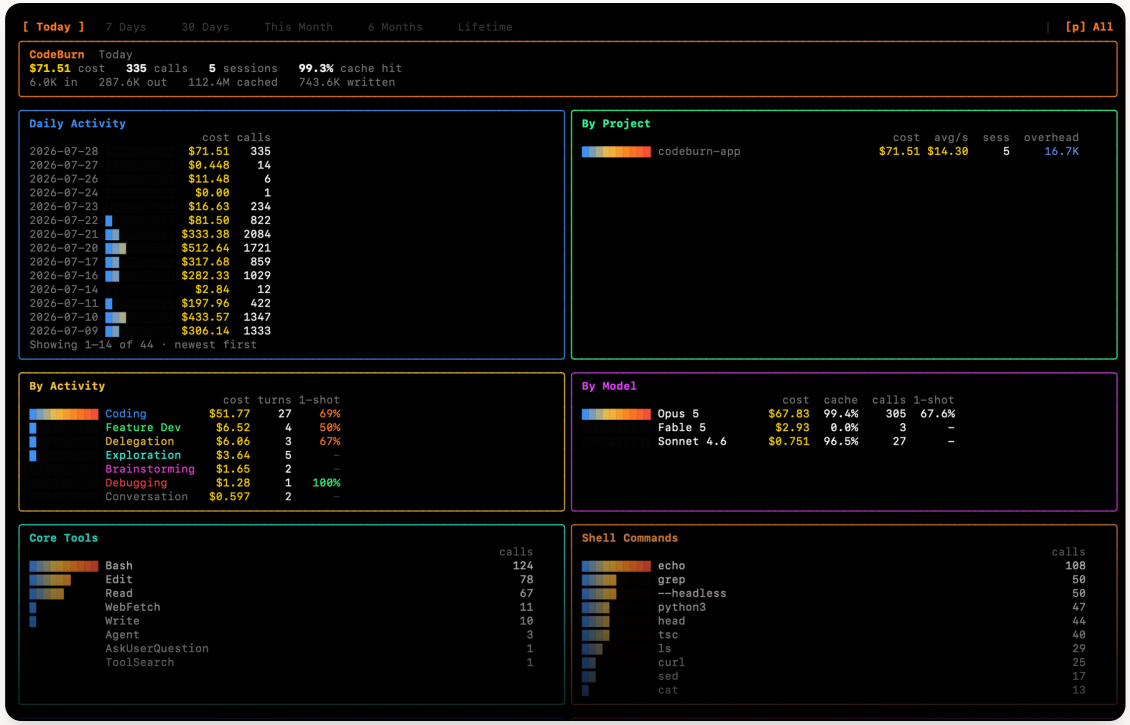
Session-Analyse-Tools unterstützen fast jeden Coding-Agenten, z. B.

- Codeburn
- Agentsview

⚠ Die *absoluten* Kosten, die diese Tools anzeigen, sind oft ungenau

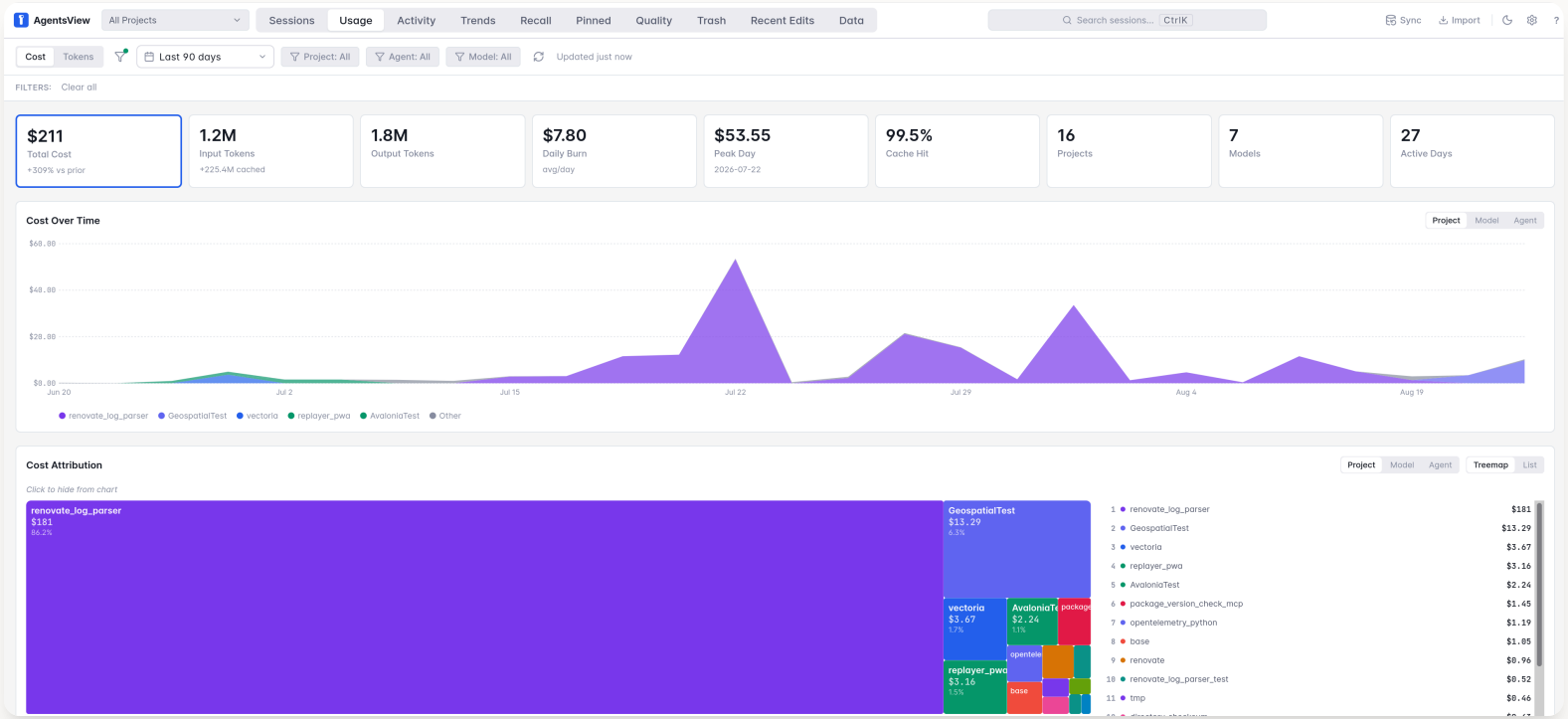
Codeburn im Terminal

Kompakte Live-Ansicht direkt in der Shell – Kosten, Tokens und Modelle auf einen Blick



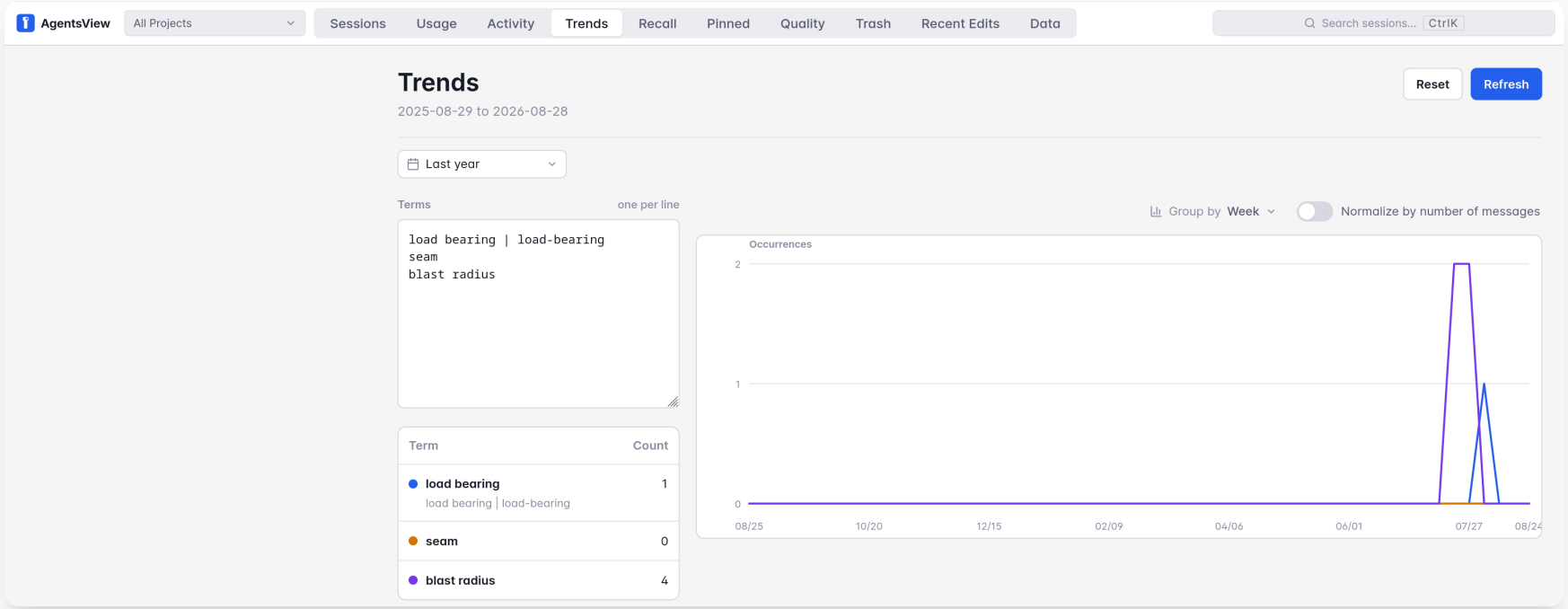
Agentsview: Usage

Verbrauch über Zeit, aufgeschlüsselt nach Modell und Projekt



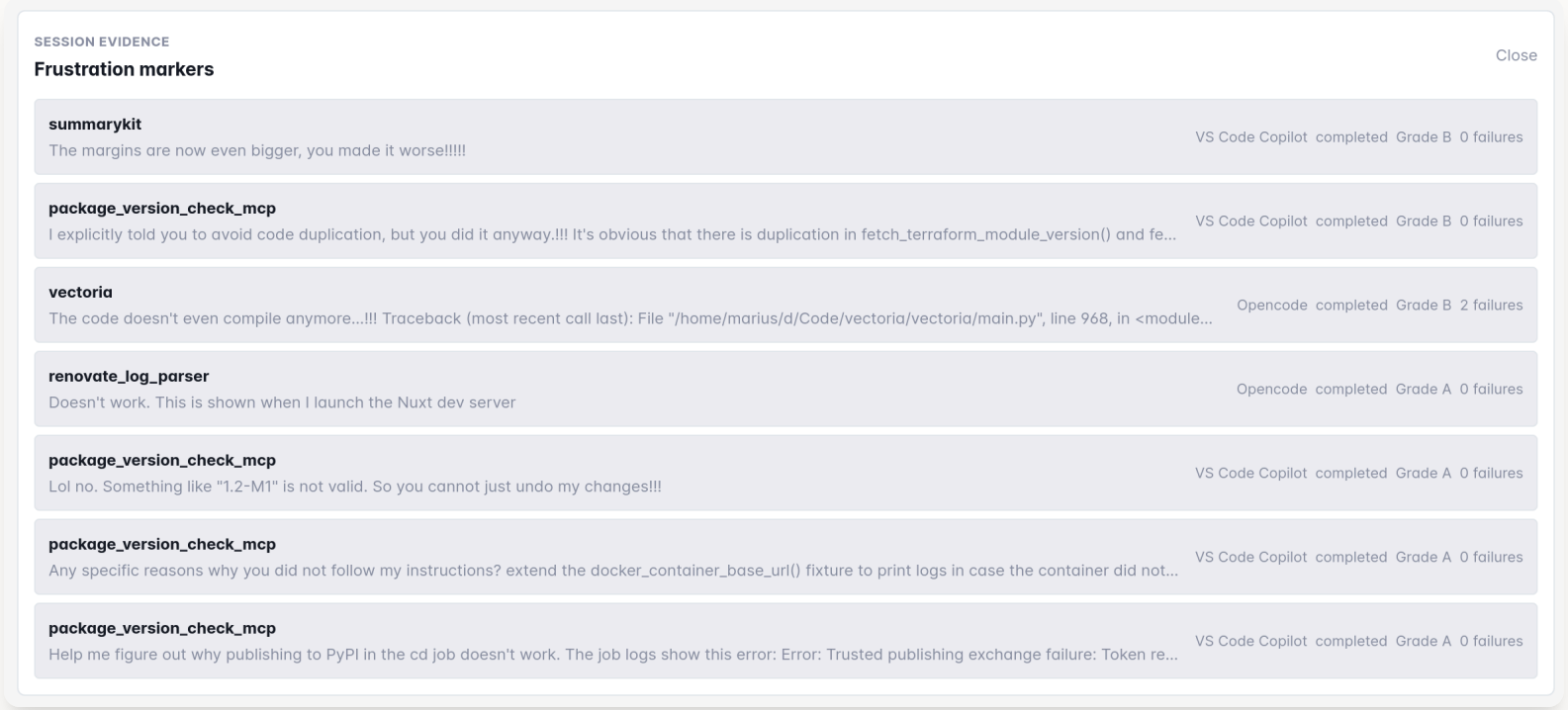
Agentsview: Trends

Trends in der Wort-Nutzung erkennen (Hallo, Claudism)



Agentsview: Frustration Markers

Erkennt Muster wie wiederholte Korrekturen – ein Hinweis auf ineffiziente Sessions



Danke! 👋

Fragen?

SCAN ME

In welche Themen wollt ihr tiefer einsteigen?