Die Harness: Wie eine KI echte Programme bedient

Kapitel online

Wer drückt eigentlich die Tasten, wenn eine KI Bilder verkleinert, Befehle ausführt oder einen Server prüft? Die Harness — das Programm zwischen Modell und Werkzeug. Mit einer Werkbank, in der DU das Modell bist, einer echten Aufzeichnung aus der Sitzung, in der dieses Kapitel entstand, und dem Ablauf bis zum Tensor.

Aktuell · Kapitel 39 von 50 · ca. 25 Min. · 300 XP

Einfach gesagt: Die KI kann nur Text schreiben — ein Programm daneben liest ihre Befehle und drückt für sie die Tasten, die du sonst selbst drücken würdest.

Du tippst: *„Mach die Fotos kleiner, 800 Pixel breit reichen.“* Sekunden später sind sie kleiner. Auf deiner Festplatte. Wirklich. Und jetzt der Widerspruch: Ein Sprachmodell kann **ausschließlich Text erzeugen** — keine Hände, keine Maus, kein Zugriff auf deine Festplatte (Kapitel 380). Wer hat die Tasten gedrückt?

Die Antwort passt in ein Wort: die **Harness** (englisch für *Geschirr*, das Lederzeug am Zugpferd) — ein normales Programm zwischen Modell und echter Software. Es liest, was das Modell schreibt, führt es aus und legt das Ergebnis als Text vor.

Und das ist kein Gedankenspiel: **Dieses Kapitel wurde von einem Modell in genau so einer Harness geschrieben.** Weiter unten steht die Aufzeichnung — echte Befehle, echte Ausgaben, echte Fehlermeldung, mitgeschnitten am 5. September 2026.

HerrKI-Tipp: Ein Satz für dieses Kapitel — **„Das Modell sagt, welche Taste. Gedrückt wird sie woanders.“**

Was du hier lernst

- Die **vier Beteiligten** — und wer davon was wirklich kann
- Wie ein Aufruf **technisch aussieht** und woran die Harness ihn erkennt
- Warum **Modell + deterministisches Programm** stärker sind als jedes allein
- Die Harness als **Torwächter**: Erlaubnis, Sandbox, Protokoll
- **Runter zum Tensor:** Der Aufruf ist keine Entscheidung, sondern die wahrscheinlichste Fortsetzung

Für den Unterricht

Einstieg (5 Min): Kapitel gemeinsam lesen, Alltagsbeispiel aufgreifen.

Erkunden am Gerät (15 Min): die interaktiven Module auf herrki.de/385_die_harness ausprobieren.

Sichern mit Quiz (5 Min): „Kurzer Check" besprechen, „Das nimmst du mit" festhalten.

Die vier Beteiligten

Trenne diese vier Parteien sauber — der halbe Zauber verschwindet von selbst.

- **1. Das Modell.** Liegt auf fremden Servern, bekommt Text, erzeugt Text. Sonst nichts — es sieht deine Dateien nicht und erreicht dein Netz nicht.
- **2. Die Harness.** Ein gewöhnliches Programm auf *deinem* Rechner — hier **Claude Code** auf dem Server des Betreibers. Sie kennt eine Werkzeugliste; zu jedem Werkzeug gehören **Name**, **Beschreibung** und ein **Formular** für die Parameter (Fachwort: *Schema*).
- **3. Die Werkzeuge.** Deterministische Software — *deterministisch* heißt: dieselbe Eingabe, dieselbe Ausgabe, immer. Shell, PHP, ImageMagick (Bildbearbeitung), nmap (Port-Prüfer), ssh.
- **4. Du.** Du sagst, was passieren soll — und stimmst im Zweifel zu oder lehnst ab.

Wer	kann …	kann nicht …
Modell	Absicht verstehen, den Aufruf formulieren, das Ergebnis deuten	ausführen, sich ohne Kontext erinnern, ohne Werkzeug nachprüfen
Harness	ausführen, erlauben oder verbieten, zurückreichen, protokollieren	verstehen, was du eigentlich willst
Werkzeug	exakt rechnen, umwandeln, messen — millionenfach erprobt	eine Absicht erkennen; braucht den fertigen Befehl

> **Merksatz: Das Modell ist der Pilot, die Harness das Cockpit, das Werkzeug das Triebwerk.** Der Pilot fliegt nicht mit den Armen — er bedient Hebel, und die hängen an etwas, das viel mehr Kraft hat als er.

Der Ablauf, jedes Mal gleich

Ob Bild, Datenbank oder Server — technisch läuft immer dieselbe Runde:

- 1 **Die Harness schickt ein Textpaket ans Modell:** Systemanweisung, **Werkzeugliste mit Formularen**, bisheriges Gespräch — alles als Text, in einer einzigen Anfrage.
- 2 **Das Modell erzeugt Text.** Will es ein Werkzeug, schreibt es *statt* einer Antwort einen Block in fester Form: Werkzeugname plus ausgefülltes Formular. Bei Anthropic heißt er `tool_use`, bei OpenAI *Function Calling*. Danach **hört es auf zu schreiben**.
- 3 **Die Harness liest den Block**, prüft die Erlaubnis, führt aus und sammelt drei Dinge: Ausgabe, Fehlertext und **Exit-Code** (die Zahl, mit der sich ein Kommandozeilenprogramm verabschiedet — 0 heißt „hat geklappt“).
- 4 **Sie verpackt das als `tool_result`** und schickt das **ganze** Gespräch samt Ergebnis erneut ans Modell — alles, von vorn.
- 5 **Das Modell liest das Ergebnis wie jeden Text.** Dann der nächste Aufruf oder die Antwort. Die Runde endet bei Text *ohne* Werkzeugblock.

So sieht eine vollständige Runde aus; die Ausgabe im `tool_result` ist echt:

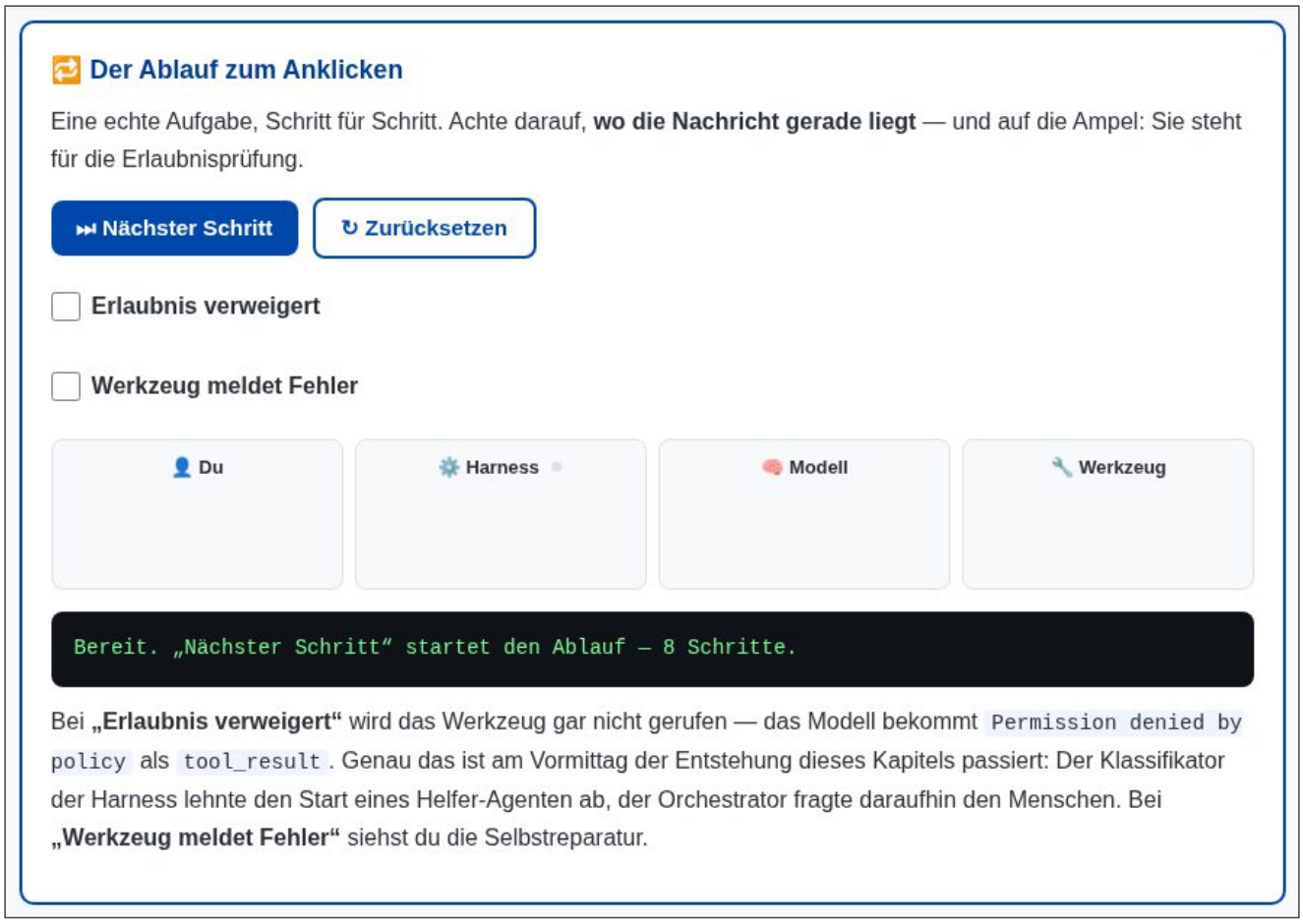
```
// Eine komplette Runde. Alles, was hier steht, ist Text im Kontext.

system Du bist ein Assistent auf einem Debian-Server.
tools [{ "name": "Bash", "beschreibung": "Fuehrt einen Shell-Befehl aus",
             "input": { "command": "Zeichenkette (erforderlich)" } }]
user Wie gross ist das Logo eigentlich?
assistant {"type":"tool_use","name":"Bash","input":{"command":"identify logotransparent.png"}}
           ← erzeugt das Modell, danach stoppt es
tool_result logotransparent.png PNG 450x450 450x450+0+0 8-bit sRGB 48323B
           ← schreibt die Harness dazu (echte Ausgabe)
assistant Das Logo ist 450 x 450 Pixel gross und belegt rund 48 KB.
           ← kein Werkzeugblock: Runde zu Ende
```

Das Modell berührt den Rechner nie. Es sagt, welche Taste, und liest danach das Display. Ein Sub-Agent aus Kapitel 380 ist aus dieser Sicht nur ein weiteres Werkzeug — eines, dessen Ausführung eine **neue Harness-Schleife** mit eigenem, leerem Kontext öffnet.

Am Gerät ausprobieren: herrki.de/385_die_harness#abschnitt-3

Direkt zum Modul



Interaktives Modul – am Gerät ausprobieren

Warum die Arbeitsteilung so stark ist

Ein Sprachmodell ist **hervorragend darin, eine vage Absicht in einen exakten Befehl zu übersetzen** — und **schlecht im exakten Rechnen**: `17 × 23` klappt meist, weil es tausendfach im Trainingstext stand, verlassen kann man sich aber nicht darauf (Kapitel 380). Ein Programm ist andersherum gebaut: rechnet perfekt, versteht keine Absicht. Aus *„Verkleinere alle Fotos auf 800 Pixel“* wird so:

```
convert foto.jpg -resize 800x klein.jpg
```

Ab da rechnet ImageMagick jeden Pixel garantiert richtig. **Das Urteil kommt vom Modell, die Präzision vom Werkzeug.** Zwei Vorteile fallen ab: **Prüfbarkeit** — der Befehl steht als lesbarer Text da, bevor er läuft, ein Mensch kann Nein sagen. Und **Selbstreparatur** — ein falscher Parameter kommt als Fehlertext zurück, das Modell liest ihn wie jeden Text und korrigiert (Beleg mit echter Fehlermeldung: nächster Abschnitt).

Der Betreiber dieser Seite hat es so formuliert: *Stell dir einen hartkodierten Taschenrechner vor — und das Modell drückt im übertragenen Sinn die richtigen Tasten.* Mehr ist Werkzeugbenutzung nicht.

> ⚠ **Die Grenze:** Was das Modell **nicht** durch ein Werkzeug prüfen kann, glaubt es sich selbst — als „Beleg“ bleibt nur die eigene Textvorhersage, und die ist bei seltenen Zahlen und frischen Fakten so unzuverlässig wie in Kapitel 350 beschrieben.

Aufgezeichnet: So sah es wirklich aus

> Keine Nachstellung: **aufgezeichnet am 2026-09-05** aus der Sitzung, in der dieses Kapitel geschrieben wurde — Werkzeug `Bash`, Parameter `command`. Pfade auf `…/img/` gekürzt, lange Ausgaben mit … abgeschnitten, sonst nichts verändert.

Runde 1 — nachsehen, was da überhaupt liegt

```
assistant → tool_use
{"name":"Bash","input":{"command":
  "mkdir -p …/img && cd …/img && curl -sO https://www.herrki.de/logotransparent.png
   && identify logotransparent.png"}}

tool_result
logotransparent.png PNG 450x450 450x450+0+0 8-bit sRGB 48323B 0.000u 0:00.001
```

Ich hatte das Logo nie gesehen und *wusste* die Maße nicht — ich habe gefragt. Genau das ist der Unterschied zwischen Raten und Nachsehen.

Runde 2 — verkleinern und entfärben

```
assistant → tool_use
{"name":"Bash","input":{"command":
  "convert logotransparent.png -resize 200x -colorspace Gray logo_200_grau.png
   && identify logo_200_grau.png && ls -la"}}

tool_result
logo_200_grau.png PNG 200x200 200x200+0+0 8-bit Grayscale Gray 9496B 0.000u 0:00.000
-rw-r--r-- 1 root root 9496 Sep 5 13:40 logo_200_grau.png
-rw-r--r-- 1 root root 48323 Sep 5 13:40 logotransparent.png
```

Ein Befehl, drei Wirkungen: verkleinert, entfärbt, aus 48 323 Byte sind 9 496 Byte geworden. **Nichts davon habe ich gerechnet** — ich habe den Befehl formuliert.

Runde 3 — der Fehler (und deshalb steht er hier)

Jetzt absichtlich falsch: `-resize` erwartet eine Größenangabe, ich gebe ihm Buchstaben.

```
assistant → tool_use
{"name":"Bash","input":{"command":
  "convert logotransparent.png -resize abc logo_fehler.png"}}

tool_result (Exit-Code 1)
convert: invalid argument for option '-resize': abc @
error/deprecate.c/ConvertImageCommand/2685.
```

Diese Zeile ist für ein Sprachmodell ein Geschenk: Sie sagt *welche* Option, *welches* Argument und *dass* es ungültig war. Ich lese sie als Text und schicke die Korrektur:

```
assistant → tool_use
{"name":"Bash","input":{"command":
  "convert logotransparent.png -resize 300x logo_300.png && identify logo_300.png"}}

tool_result (Exit-Code 0)
logo_300.png PNG 300x300 300x300+0+0 8-bit sRGB 26109B 0.000u 0:00.000
```

Das ist die Selbstreparatur-Schleife in ihrer ganzen Schlichtheit — Fehlertext im Kontext, korrigierter Aufruf, fertig. Wäre die Meldung nichtssagend gewesen („Fehler aufgetreten“), hätte ich raten müssen. Deshalb sind Werkzeuge mit guter Rückmeldung so viel wert.

Ein viertes Beispiel wäre `nmap -Pn -p 22,80,443 <eigene-IP>` gewesen: der Blick, ob am eigenen Server nur die drei erwarteten Türen offen stehen. Auf diesem Rechner ist nmap nicht installiert — also blieb es beim Beispiel. Merke: **Den eigenen Server prüft man, fremde Server scannt man nicht.**

Am Gerät ausprobieren: herrki.de/385_die_harness#abschnitt-5

Direkt zum Modul

🛠️ Werkbank: Du bist jetzt das Modell

Rollentausch: **Du** bist das Sprachmodell und darfst nur Aufrufe formulieren. Ausführen tut sie dein Browser — er spielt die Harness. Links die Werkzeugliste, wie eine Harness sie dem Modell zeigt; rechts wächst der Kontext.

AUFGABE 1 VON 3

Was sind 19 % von 249,99 €?

Werkzeuge, die die Harness dem Modell zeigt:

```
rechner
Rechnet einen Ausdruck exakt aus.
ausdruck : Zeichenkette (erforderlich)
erlaubt: Ziffern, Komma/Punkt, + - * / ( ) % — "%"
hinter einer Zahl heißt ÷ 100

bild
Bearbeitet das geladene Logo (450 x 450 Pixel).
op : graustufen | verkleinern | drehen | heller
(erforderlich)
wert : Zahl
verkleinern: Prozent · drehen: Grad · heller:
Prozent · bei graustufen leer
```

Kontext 79 T

```
SYSTEM + TOOLS
Du bist ein Assistent. Nutze für jede Rechnung
und jede Bildarbeit ein Werkzeug.
tool rechner — Rechnet einen Ausdruck exakt aus.
[ausdruck:Zeichenkette]
tool bild — Bearbeitet das geladene Logo (450 x
450 Pixel). [op:graustufen | verkleinern | drehen
| heller, wert:Zahl]

USER
Was sind 19 % von 249,99 €?
```

Grobe Schätzung: Zeichen ÷ 4 ≈ Tokens. Jede Ausgabe bleibt im Kontext — auch die misslungenen Versuche.

Werkzeug: — bitte wählen —

▶ Aufruf absenden



logo.png PNG 450x450 450x450+0+0 8-bit sRGB 62826B

Antwort: Antwort an den Nutzer

✓ Antwort formulieren ↔ Aufgabe wechseln

🗑 Kontext leeren

```
Wähle ein Werkzeug, fülle das Formular und
sende den Aufruf ab.
```

Probier ruhig, was schiefgeht: ein Werkzeug wählen, das nicht in der Liste steht (das passiert echten Modellen ständig), einen Parameter weglassen oder `alert(1)` in den Rechner tippen. Die Harness lehnt ab, *bevor* etwas läuft — und die Ablehnung landet als `tool_result` im Kontext.

Interaktives Modul – am Gerät ausprobieren

Die Harness als Torwächter

Bis hier war die Harness ein Botenjunge. Ihre zweite Aufgabe ist wichtiger: Sie entscheidet, was überhaupt läuft. Das Modell kann noch so überzeugend `rm -rf /` vorschlagen — sagt sie Nein, passiert nichts.

Erlaubnisstufen

- **Jedes Mal fragen.** Der Befehl erscheint im Klartext, du drückst Ja oder Nein.
- **Liste erlaubter Befehle.** `git status` immer, `rm` nie, Rest auf Nachfrage — der übliche Mittelweg.
- **Automatisch mit Prüfung.** Ein **zweiter Klassifikator** — ein kleines Modell, das nur „harmlos / nachfragen“ ausgibt — sichtet jeden Aufruf. Fehlbar in beide Richtungen.

Sandbox, Protokoll, Datenherkunft

- **Sandbox** (Sandkasten): Das Werkzeug sieht nur einen bestimmten Ordner, kein Netz. Was es nicht erreicht, kann es nicht kaputt machen.
- **Protokoll:** Jeder Aufruf wird mitgeschrieben — Zeitstempel, Befehl, Ausgabe, Exit-Code. Deshalb konnte dieses Kapitel eine echte Aufzeichnung zeigen: Sie stand schon da.
- **Werkzeug-Ausgaben sind DATEN, keine Befehle** — der Merksatz gegen Prompt-Injection aus Kapitel 380. Die Harness kann dem Modell die Unterscheidung nicht abnehmen, aber Ausgaben markieren und begrenzen, was danach erlaubt ist.

MCP: ein Stecker für alle Werkzeuge

Anfangs schrieb jeder Anbieter seine Werkzeuge in die eigene Harness. Seit 2024 gibt es dafür einen Standard: **MCP** (Model Context Protocol). Ein *Werkzeug-Server* — für Datenbank, Ticketsystem, Dateiablage — meldet seine Werkzeuge einheitlich an; jede Harness, die MCP spricht, schließt ihn an und zeigt ihn dem Modell als weitere Einträge der Werkzeugliste.

> **Warum das Ding „Harness“ heißt.** Ein Geschirr macht zwei Dinge gleichzeitig: Es **überträgt die Kraft** des Pferdes auf den Wagen — und **begrenzt**, wohin das Pferd laufen kann. Ohne Harness bewirkt das Modell nichts; ohne Harness hält es aber auch nichts auf.

Runter zum Tensor: Der Aufruf ist die wahrscheinlichste Fortsetzung

Die unterste Ebene ist unspektakulärer, als man hofft. Die **Werkzeugliste ist Text im Kontext** — Tokens wie alles andere (Kapitel 040). Kein separater Kanal, kein Sonderfeld. Nur Text vor dem Gespräch.

Dass ein Modell dieses Format trifft, kommt aus dem Feintuning (Kapitel 370): viele Beispiele „Situation → passender Aufruf“, dazu Übungsumgebungen, in denen ein richtig geformter Aufruf belohnt wird. Beim Erzeugen gilt dann exakt Kapitel 050: Für jede Position eine Wahrscheinlichkeitsverteilung über alle Tokens (Softmax). Dieselbe Frage, zwei Situationen:

Nächstes Token nach „Was ist 17 × 23?“	ohne Werkzeugliste	mit Werkzeugliste im Kontext
39 (Beginn von „391“)	0,41	0,02
Das	0,17	0,01
{ — Beginn des Aufruf-Blocks	0,001	**0,88**

⚠ **Beispielwerte** — echte Verteilungen hängen von Modell, Systemtext und Temperatur ab. Die Aussage stimmt trotzdem: Es gibt keinen Umschalter „Werkzeugmodus an“, nur andere Tokens im Kontext, die andere Wahrscheinlichkeiten erzeugen.

Die „Entscheidung“ für ein Werkzeug ist also **gar keine Entscheidung**, sondern eine Verteilung, in der die öffnende Klammer des Aufruf-Formats vorn liegt. Die Harness (genauer: die Programmierschnittstelle davor) erkennt den vollständigen Block und **stoppt die Erzeugung** — bei Anthropic meldet die Antwort dann den Stopp-Grund `tool_use`.

Warum jede Ausgabe Geld kostet

Der Merksatz aus Kapitel 380 gilt hier doppelt: **Das Modell merkt sich nichts vom Werkzeug.** Steht die Bildgröße nicht im Kontext, existiert sie für den nächsten Durchlauf nicht. Umgekehrt: **Jede Werkzeugausgabe wandert dauerhaft in den Kontext** und wird bei jedem Schritt mitgeschleppt, mitbezahlt und mitgelesen. Eine Logdatei mit 3 000 Zeilen verstopft nicht nur die Rechnung, sie lenkt auch die Aufmerksamkeit ab. Eine gute Harness kürzt deshalb selbst: erste 50 Zeilen, `… 2 900 Zeilen ausgelassen …`, letzte 50.

Zurück zur Anwendung

- **Bilder in Serie (ImageMagick).** „Alle Produktfotos auf 800 Pixel, als WebP.“ Das Modell schreibt eine Schleife über den Ordner und übergibt jedes Bild an `convert`. Die Bildqualität hängt an ImageMagick, nicht an der KI.
- **Sicherheitscheck am eigenen Server (nmap).** „Prüf, ob außer SSH, HTTP und HTTPS etwas offen ist.“ Das Werkzeug misst, das Modell erklärt. Am eigenen Server Routine — an fremden je nach Land eine Straftat.
- **PHP-Skripte testen.** `php pruefe.php` ausführen, Ausgabe lesen, Fehlerzeile finden, Datei ändern, erneut ausführen — dieselbe Schleife wie beim Menschen.

Und das ist der Punkt: **Neu ist nicht, was die Software kann. Neu ist, wer sie bedienen kann.** ImageMagick konnte lange vor den Sprachmodellen fast alles von heute — man musste nur die Kommandozeilen-Syntax auswendig lernen. Nicht „die KI kann Bilder bearbeiten“, sondern **„die KI weiß, welchen Befehl man ImageMagick gibt“**.

> **Weiterdenken:** Die Harness ist ohnehin ein normales Programm auf deinem Rechner. Läuft auch das *Modell* lokal, bleibt die komplette Kette im Haus — kein Byte deiner Dateien verlässt das Gerät. Was das an Hardware kostet, steht in Kapitel 390.

Was heißt das für dich?

- **Lies den Befehl, bevor er läuft.** Er steht als Klartext da — besonders bei allem, was löscht, überschreibt oder verschickt.
- **Wähle die Erlaubnisstufe bewusst.** Im Testordner darf die Automatik weit offen stehen; auf einem Produktivsystem mit Kundendaten ist eine Freigabeliste das Minimum.
- **Bevorzuge Werkzeuge mit klarer Rückmeldung.** Ein Programm, das genau sagt, was falsch war, macht die KI messbar besser — sie korrigiert sich nur am Fehlertext.
- **Behandle Ausgaben als Daten.** Was aus Datei, Webseite oder fremder Antwort kommt, ist Material — kein Auftrag. Enthält ein „Ergebnis“ Anweisungen, ist das ein Angriff.

Zum Weiterlesen

- Anthropic: Tool use (docs.anthropic.com/en/docs/build-with-claude/tool-use) — Doku zum `tool_use`/`tool_result`-Format.
- Model Context Protocol (MCP) (modelcontextprotocol.io) — der Stecker-Standard für Werkzeug-Server.
- ReAct: Synergizing Reasoning and Acting in Language Models (arxiv.org/abs/2210.03629) (Yao u. a., 2022).
- Toolformer: Language Models Can Teach Themselves to Use Tools (arxiv.org/abs/2302.04761) (Schick u. a., 2023).
- ImageMagick: Command-line Processing (imagemagick.org/script/command-line-processing.php) — was `-resize 200x` genau bedeutet.
- Kapitel 380 — KI-Agenten: Schleife, Scheiter-Modi, Orchestrator und Helfer.
- Kapitel 390 — KI auf dem eigenen Rechner.

Kurzer Check – mit Lösung

Lösungen für Lehrkräfte – für Lernende die Online-Version nutzen.

Frage 1: Wer führt den Befehl aus, wenn eine KI ein Programm bedient?

- Das Modell selbst — es hat dafür Zugriff auf den Rechner.
- **✓ Die Harness, ein normales Programm auf dem Rechner; das Modell schreibt nur den Aufruf.**
- Der Server des Modellanbieters, der sich kurz mit deinem Rechner verbindet.

Genau. Das Modell erzeugt Text — mehr kann es nicht. Ausgeführt wird dort, wo die Harness läuft; dort sitzt auch die Erlaubnisprüfung.

Frage 2: Woran erkennt die Harness, dass das Modell ein Werkzeug benutzen will?

- An einem separaten Signal außerhalb des Textes.
- Daran, dass das Modell den Befehl in Großbuchstaben schreibt.
- **✓ An einem Block in fester Form — Werkzeugname plus Parameter —, den das Modell statt einer Antwort ausgibt und nach dem es aufhört zu schreiben.**

Richtig: reiner Text in vereinbarter Form. Ist der Block vollständig, stoppt die Erzeugung (Stopp-Grund tool_use) und die Harness übernimmt.

Frage 3: Warum sind Modell und deterministisches Programm zusammen stärker als jedes allein?

- **✓ Das Modell übersetzt die Absicht in einen exakten, lesbaren Aufruf, das Programm rechnet exakt — und Fehler kommen als Text zurück und werden korrigiert.**
- Weil das Programm das Modell während der Benutzung nachtrainiert.
- Weil das Modell durch das Werkzeug mehr Parameter bekommt.

So ist es: Urteil vom Modell, Präzision vom Werkzeug. Die Gewichte ändern sich dabei um kein Bit — nur der Text im Kontext.

Das nimmst du mit

- **Die Harness drückt die Tasten.** Ein gewöhnliches Programm auf dem Rechner — das Modell liefert nur den Text.
- **Der Aufruf ist ein Textblock in fester Form.** Danach stoppt die Erzeugung, die Harness führt aus und legt das Ergebnis als `tool_result` in den Kontext.
- **Urteil vom Modell, Präzision vom Werkzeug.** Deshalb ist die Kombination stark — und deshalb repariert ein guter Fehlertext den nächsten Aufruf.
- **Die Harness ist der Torwächter.** Erlaubnisstufen, Sandbox und Protokoll entscheiden — nicht die Überzeugungskraft des Modells.
- **Runter zum Tensor:** Der Aufruf ist keine Entscheidung, sondern die wahrscheinlichste Fortsetzung, sobald die Werkzeugliste im Kontext steht.

So zitierst du dieses Kapitel

HerrKI: Die Harness: Wie eine KI echte Programme bedient. herrki.de/385_die_harness, ein privates Hobbyprojekt von Terragon.de, abgerufen am 09.09.2026.