

AI Engineering

Vom Prompt zum Agenten: ReAct, Tool-Use und die Sicherheitsgurte

Ein Agent ist kein Prompt mit Werkzeugen, sondern eine kontrollierte Schleife aus Denken, Handeln und Beobachten.

Band 1 · Kapitel 18

Öffentliche Leseprobe · Auszug (§1 „Das Problem“ bis §4 „Architektur“)

Sebahattin Özata · Diplom-Informatiker

OMEX Press

Aus Kapitel 18 in Band 1: der Moment, in dem ein Modell nicht mehr nur antwortet, sondern handelt. Kapitelverweise beziehen sich auf das vollständige dreibändige Werk.

Kapitel 18 — Vom Prompt zum Agenten: ReAct, Tool-Use und die Sicherheitsgurte

Ein-Satz-Kern: Ein Agent entsteht, sobald ein Modell nicht nur denkt, sondern auch *handeln* und das Ergebnis seiner Handlung *beobachten* darf — und genau diese Freiheit macht ihn erst nützlich und sofort gefährlich, weshalb jeder produktive Agent von harten, extern erzwungenen Grenzen umgeben sein muss.

Dies ist das erste Kapitel von Teil V. Bis hier hat unser Modell nur *geredet*: es hat Text bekommen und Text zurückgegeben. In diesem Kapitel geben wir ihm zum ersten Mal **Hände** — die Fähigkeit, in der Welt etwas zu tun. Das ist der Moment, in dem aus einem Chatbot ein *Agent* wird. Und es ist der Moment, in dem Ingenieurskunst wichtiger wird als Modellqualität.

Lernziele

Nach diesem Kapitel kann der Leser ...

- **(Recall)** die Bausteine der **ReAct**-Schleife (*Reasoning + Acting*; Thought, Action, Observation) und die Ebenen der Sicherheits-Pyramide benennen.
- **(Understanding)** begründen, warum Chain-of-Thought ohne Werkzeuge kein ReAct ist und warum ein Guard in der Runtime stärker ist als einer im Prompt.
- **(Understanding)** die vier Ebenen **schema-gültig** → **autorisiert** → **ausgeführt** → **vertrauenswürdig** trennen und begründen, warum ein erfolgreiches Tool-Resultat nicht automatisch wahr ist (**Tool-Resultat** ≠ **Wahrheit**).
- **(Application)** für eine gegebene Aufgabe zwischen Chain-of-Thought, ReAct und fester Pipeline wählen (Ü1, Ü4).
- **(Analysis)** in einem Agentenentwurf die fehlenden Guards finden, die einen Ausreißer zulassen (Ü3).
- **(Evaluation)** den Trade-off zwischen Agent-Freiheit und Vorhersehbarkeit bewerten und verteidigen (Ü4).

- **(Creation)** eine vollständige Guard-Konfiguration inklusive Human-in-the-Loop für einen handelnden Agenten entwerfen (Ü2).

1. Das Problem

Stell dir eine ganz normale Anfrage an ein Steuer-Assistenzsystem vor:

„Ist die Rechnung von Lieferant Meier über 4.900 Euro schon bezahlt, und wenn nein, wann ist sie fällig?“

Ein reines Sprachmodell — egal wie gut — **kann diese Frage nicht beantworten**. Nicht, weil es zu dumm wäre, sondern weil die Antwort **nicht in seinem Kopf** steht. Sie steht in einer Datenbank, die es beim Training nie gesehen hat. Das Modell weiß nichts über Lieferant Meier, nichts über diese Rechnung, nichts über den heutigen Zahlungsstatus.

Ein Modell hat genau zwei Wissensquellen:

1. Was es beim **Training** gelernt hat (mit einem Stichtag in der Vergangenheit).
2. Was in **diesem Moment** in seinem Kontext steht (der Prompt).

Alles andere — Live-Daten, Datenbanken, Web-Inhalte, Benutzerkonten, das aktuelle Wetter, der heutige Kontostand — ist für das Modell schlicht nicht existent.

Das ist das Kernproblem dieses Kapitels: **Wie bringt man ein Modell dazu, Dinge zu tun, die es allein nicht kann — nachschauen, rechnen, suchen, buchen — und mit dem Ergebnis sinnvoll weiterzuarbeiten?**

Und sobald wir das gelöst haben, entsteht sofort ein zweites, gefährlicheres Problem: **Ein handlungsfähiges Modell kann auch Schaden anrichten**. Es kann in einer Endlosschleife 3.000 Tickets erzeugen, in Minuten vierstellige Kosten produzieren oder eine E-Mail an alle Kontakte schicken. Wer einem Modell Hände gibt, muss ihm im selben Atemzug Sicherheitsgurte anlegen.

Dieses Kapitel behandelt beide Hälften: **erst die Hände (ReAct), dann die Gurte (Agent-Sicherheit)**. Sie gehören untrennbar zusammen.



Welche Modellgrenze adressiert dieses Kapitel? Ein Sprachmodell hat **keinen Zugriff auf die Außenwelt** — es kennt nur sein Trainingswissen und den aktuellen Kontext. Und es hat **keinen verlässlichen Kostensinn, kein Zeitgefühl und keine sichere Selbstbegrenzung**. Beide Hälften des Kapitels folgen daraus: Tool-Nutzung öffnet den Weltzugang — und die Grenzen dieses Zugangs muss die **Runtime** erzwingen, nicht das Modell.

2. Intuition

Die erste Hälfte: der Unterschied zwischen Nachdenken und Handeln

Ein Bild aus dem Büroalltag.

Chain-of-Thought — der Mitarbeiter, der nur nachdenkt:

Du fragst einen Kollegen: "Was kosten 3,5 Kilo Äpfel bei 2,40 € pro Kilo?"

Er lehnt sich zurück, überlegt laut:

"2,40 mal 3 sind 7,20 ... mal 0,5 sind 1,20 ... zusammen 8,40 €."

Er hat nichts nachgeschlagen. Alles lief in seinem Kopf ab.

Das funktioniert wunderbar — **solange die Antwort in seinem Kopf verfügbar ist.**

ReAct — der Mitarbeiter, der arbeitet:

Du fragst denselben Kollegen: "Ist die Meier-Rechnung schon bezahlt?"

Jetzt kann er nicht einfach nachdenken. Er muss handeln:

denkt: "Ich muss im Buchhaltungssystem nachsehen."

handelt: öffnet das System, sucht die Rechnung ←

Aktion

beobachtet: "Status: offen, fällig am 15." ←

Beobachtung

denkt: "Also ist sie nicht bezahlt, fällig am 15." ←

weiterdenken

antwortet: "Nein, offen, fällig am 15."

Der entscheidende Unterschied ist nicht „mehr Denken“. Es ist die **Schleife**: denken → handeln → **das Ergebnis lesen** → weiterdenken. Das Zurücklesen des Ergebnisses ist das Herzstück. Ohne Werkzeug (das Buchhaltungssystem) ist ReAct gar nicht möglich — dann bleibt nur Nachdenken übrig.

Merke: Chain-of-Thought ist Denken über den eigenen Kopf. ReAct ist Denken, das mit Handeln und Beobachten verschränkt ist. Ohne Werkzeuge hat man kein ReAct, sondern nur Chain-of-Thought.

Die zweite Hälfte: warum der Autopilot einen Sicherheitsgurt braucht

Ein zweites Bild — jetzt zur Gefahr.

Ein Agent ist wie ein Autopilot im Flugzeug.

Er ist enorm nützlich: er fliegt stundenlang präzise, ohne zu ermüden.

Aber ein Autopilot ohne Grenzen ist ein Albtraum:

- Er hat kein Gefühl dafür, wie viel Treibstoff er verbrennt.
- Er merkt nicht, dass er seit einer Stunde im Kreis fliegt.
- Er hat keinen eingebauten Not-Aus.

Deshalb baut kein Ingenieur einen Autopiloten OHNE:

- Treibstoff-Warnung (Budget),
- maximale Flugzeit (Iterations-/Zeitgrenze),
- einen roten Knopf im Cockpit (Kill-Switch),
- und für die Landung: einen Menschen, der zustimmt (Human-in-the-Loop).

Der Denkfehler, der immer wieder passiert: *„Der Agent soll selbst entscheiden, wann er fertig ist.“* Das klingt elegant und ist gefährlich. **Ein Modell hat keinen Kostensinn, kein Zeitgefühl, und merkt oft nicht, dass es in einer Schleife steckt.** Wenn das Modell selbst der Wächter ist, gibt es keinen Wächter.

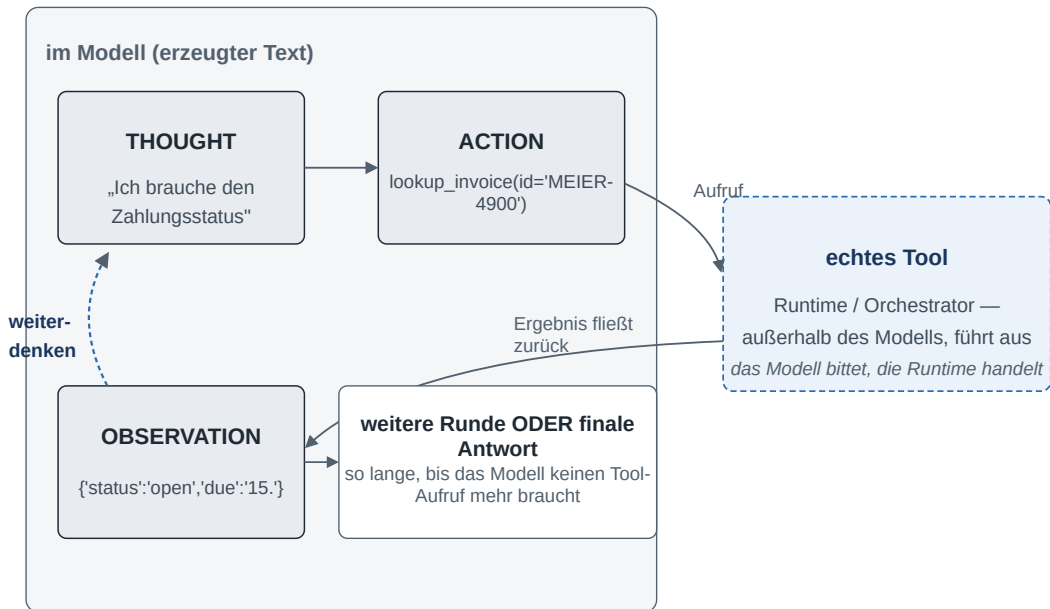
3. Technische Grundlagen

ReAct als Schleife

„ReAct“ steht für **Reasoning + Acting** (Yao et al., Preprint 2022 / ICLR 2023, arXiv:2210.03629) . Es ist ein **Prompt-/Interaktionsmuster** — kein Protokoll und kein Standard — das das Modell in eine Schleife zwingt, die abwechselnd aus drei Bausteinen besteht:

Die ReAct-Schleife — denken, handeln, das Ergebnis zurücklesen

Das Herzstück ist die Rückkopplung: die Beobachtung fließt zurück ins nächste Denken — kein linearer Ablauf



Die Schleife lesen — warum die Rückkopplung der Kern ist

Das Modell erzeugt **THOUGHT** und **ACTION** als Text; ein Stück Software **außerhalb** des Modells führt das Tool aus und schreibt das Ergebnis als **OBSERVATION** zurück. Der gestrichelte Pfeil zurück zu **THOUGHT** ist das Herzstück: ReAct ist ein **Loop**,

kein linearer Ablauf. Ohne Werkzeug bleibt nur Nachdenken übrig — dann ist es Chain-of-Thought, nicht ReAct (T1).

Weil das Modell nur **bittet** und die Runtime **handelt**, sitzt die Kontrolle über Tool-Ausführung außerhalb des Modells — genau dort setzen später die Guards an (D2, D4). Und: die **OBSERVATION** ist ein Signal, keine Wahrheit (T2).

D1 · Die ReAct-Schleife

Wichtig: Das Modell generiert **THOUGHT** und **ACTION** als **Text**. Ein Stück Software außerhalb des Modells — der **Agent-Runtime** oder **Orchestrator** — erkennt die **ACTION**, führt das echte Tool aus, und schreibt dessen Ergebnis als **OBSERVATION** zurück in den Kontext. Dann läuft das Modell erneut. Das Modell selbst „ruft“ also nichts auf — es **bittet** um einen Aufruf, und die Runtime führt ihn aus. Diese Trennung ist später sicherheitskritisch.

Function Calling — die moderne Form der „Action“

In der Praxis formulieren Modelle ihre ACTION heute meist als **strukturierten Tool-Aufruf** (Function Calling): Das Modell bekommt eine Liste verfügbarer Tools mit einem Schema (meist JSON-Schema; JSON = *JavaScript Object Notation*), und es antwortet nicht mit Prosa, sondern mit einem Objekt der Form `{"tool": "lookup_invoice", "args": {"id": "MEIER-4900"}}`. Die Runtime prüft das gegen das Schema, führt aus, gibt das Ergebnis zurück. Das ist dieselbe ReAct-Idee, nur mit einem maschinenlesbaren Aktionsformat statt frei geparstem Text. **Wichtig:** schema-gültig heißt nur *formal korrekt geformt* — nicht *erlaubt*, nicht *ausgeführt* und schon gar nicht *wahr* (die vier Ebenen trennt T2). Dass Modelle das Tool-Aufrufen überhaupt erlernen können, zeigt die Forschung — etwa **Toolformer** (Schick et al. 2023, [Studienbefund] : Modelle bringen sich selbst bei, wann ein Tool zu rufen ist) und **Gorilla** (Patil et al. 2023, [Studienbefund] : zuverlässige API-Aufrufe — API = *Application Programming Interface* —, auch wenn sich Schnittstellen ändern — ein Grund, warum **Tool-Verträge** ein eigenes Thema sind, K20).

Wo Chain-of-Thought aufhört und ReAct anfängt

T1 · Chain-of-Thought vs. ReAct

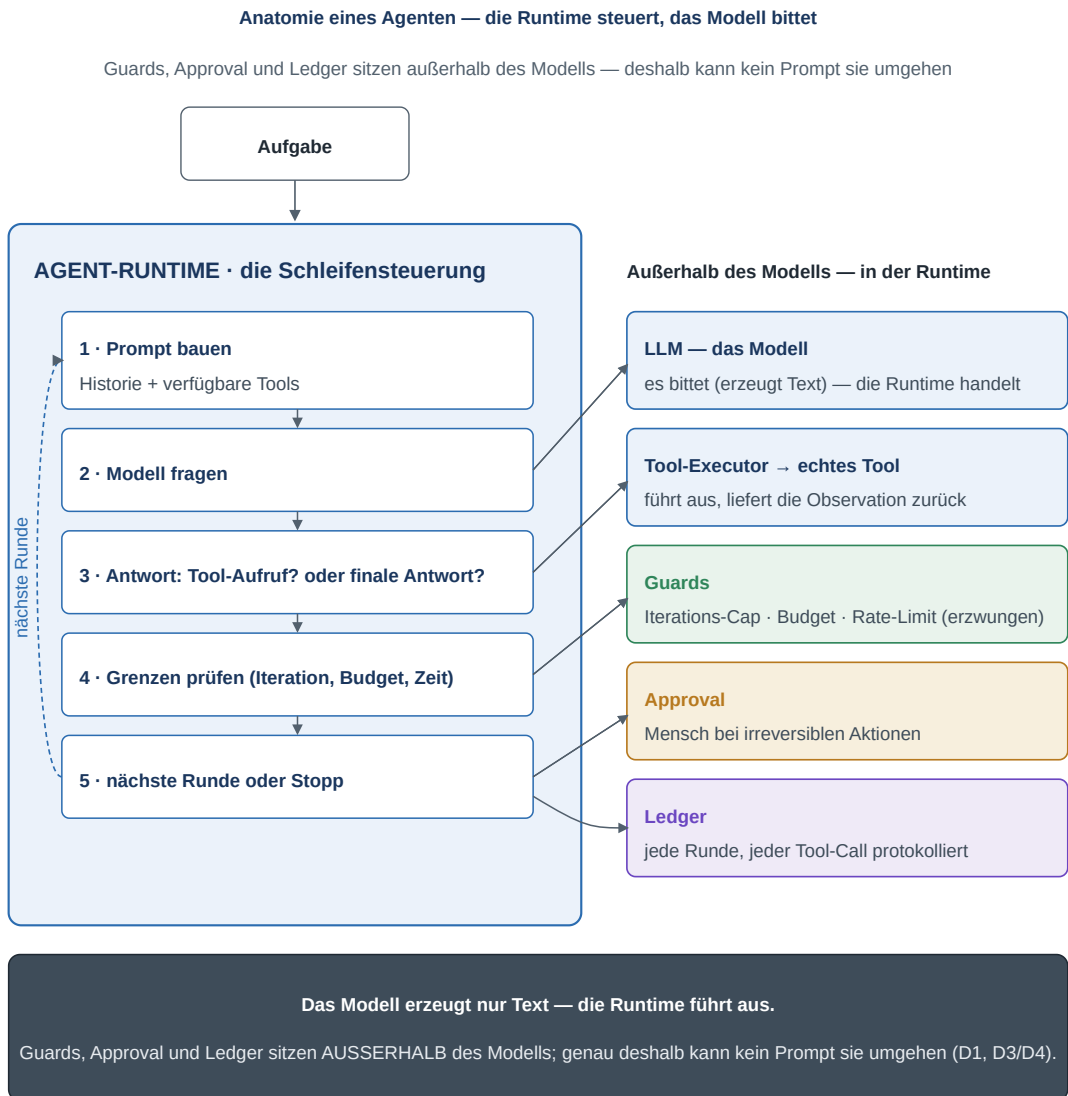
	Chain-of-Thought	ReAct
Zwischenschritte?	ja	ja
Zugriff auf die Außenwelt?	nein	ja (Tools)
Braucht Werkzeuge?	nein	ja — ohne Tools kein ReAct
Gut für	Rechnen, Logik, reines Wissen	Live-Daten, DB, Suche, Aktionen
Risiko	gering (nur Text)	hoch (kann handeln)

Ein häufiges Missverständnis lautet „ist doch dasselbe, beide denken Schritt für Schritt“. Die genaueste Korrektur lautet: *Chain-of-Thought ist Reasoning ausschließlich über den eigenen Zustand des Modells; ReAct verschränkt Reasoning mit Tool-Aufrufen und Beobachtungen — ohne Tools hat man Chain-of-Thought, nicht ReAct.*

4. Architektur

Die anatomische Zeichnung eines Agenten

Ein Agent ist kein einzelnes Ding, sondern eine kleine Fabrik. Hier die Komponenten und wie sie zusammenspielen:



D2 · Anatomie eines Agenten

Fünf Bauteile lohnt es sich zu unterscheiden:

- **Agent-Runtime** — steuert die Schleife (Prompt bauen → Modell → Tool → wiederholen).
- **Tool-Executor** — führt die tatsächlichen Aktionen aus (HTTP-Call — *HyperText Transfer Protocol* —, DB-Query, Rechnung).
- **Guards** — die extern erzwungenen Grenzen (Iterations-Cap, Budget, Zeit, Rate-Limit).
- **Approval** — der Human-in-the-Loop für gefährliche Aktionen.
- **Ledger** — das lückenlose Protokoll jeder Runde und jedes Tool-Aufrufs.

Die entscheidende architektonische Einsicht: Die Guards, das Approval und das Ledger sitzen **außerhalb** des Modells, in der Runtime. Sie sind genau deshalb wirksam, weil das Modell sie nicht umgehen, nicht überstimmen und nicht „vergessen“ kann. Ein Guard, der im Prompt steht („bitte nicht mehr als 10 Runden“), ist eine Bitte. Ein Guard, der in der Runtime als `if (runde > 10) stop()` steht, ist ein Gesetz.

Vier Ebenen, die man nicht verwechseln darf

Zwischen „das Modell hat ein Tool gewählt“ und „ich darf dem Ergebnis glauben“ liegen **vier getrennte Ebenen**. Jede beweist etwas anderes — und keine beweist die nächste.

T2 · Vier Ebenen eines Tool-Aufrufs — was jede beweist und was NICHT

Ebene	Beweist	Beweist NICHT
schema-gültig	die Argumente sind formal korrekt geformt (K12)	dass der Aufruf erlaubt, sinnvoll oder korrekt ist
autorisiert	Policy/Mandant/Region/Rechte erlauben diesen Aufruf	dass er ausgeführt wurde oder erfolgreich war
ausgeführt	das Tool lief und lieferte ein Ergebnis	dass das Ergebnis aktuell, vollständig oder wahr ist
vertrauenswürdiges Ergebnis	— bleibt eine offene Frage	wird durch die drei Ebenen darüber nicht eingelöst

◆ **Architekturregel — Tool-Resultat $\neq$ Wahrheit.** Ein erfolgreicher Tool-Aufruf beweist **Ausführung** und ggf. **Herkunft** (Provenienz) — nicht die **Wahrheit** der Aussage, die das Modell daraus formuliert. Ein Tool-Ergebnis kann **veraltet, unvollständig, manipuliert, unbefugt oder falsch interpretiert** sein. Tool-Nutzung erweitert **Handlungsraum und Nachvollziehbarkeit**, nicht **epistemische Gewissheit**. Grounding und Claim-Support bleiben die Grenze aus K15; Tool-Nutzung beseitigt Halluzination nicht (K16).

Markierungen in dieser Leseprobe

Einzelne Sätze tragen eine Markierung, die sagt, woher die Aussage stammt und wie belastbar sie ist. Hier stehen nur die, die in dieser Leseprobe vorkommen.

Markierung	Bedeutung
[Studienbefund]	Ergebnis einer einzelnen Studie oder eines Berichts, datiert — kein Konsens.