Das KI-Build-Playbook

Software, Agenten und Orchestrierungen in der richtigen Reihenfolge bauen — für deutlich bessere Ergebnisse

Eine Praxis-Checkliste von Rehm KI Consulting

Version 1.0 · Erste Ausgabe · Juli 2026

Die meisten schlechten Ergebnisse beim KI-gestützten Bauen kommen nicht von einem schwachen Modell. Sie entstehen dadurch, dass die Schritte in der falschen Reihenfolge ausgeführt werden — zu früh mit dem Programmieren zu beginnen, bevor das Problem klar umrissen ist; eine KI ihre eigene Arbeit bewerten zu lassen; an echten Nutzern zu testen statt vor ihnen.

Dies ist die Reihenfolge, die funktioniert. Arbeiten Sie sie von oben nach unten ab. Jeder Schritt nennt **was** zu tun ist, **warum** es wichtig ist und **wie „gut gemacht" aussieht** — damit Sie wissen, wann Sie weitergehen können.

> **So lesen Sie dies:** Der Haupttext funktioniert für alle, die einen KI-Build steuern. Mit **[Entwickler]** markierte Zeilen liefern die technischen Details für Programmierer. Überspringen Sie sie, wenn sie nicht für Sie sind.

Schritt 0 — Das eigentliche Problem umreißen

Was: Bringen Sie die Idee zuerst auf ein einziges, in sich stimmiges Ideendokument. Schreiben Sie das Ziel auf, Ihre Absichten und konkrete Beispiele dafür, wie **Erfolg** und **Misserfolg** aussehen.

Warum es wichtig ist: Die gestellte Anfrage ist ein Ausgangspunkt, keine Spezifikation. Die meisten verschwendeten Projekte lösen das falsche Problem — nur eben gut. Eine Stunde hier spart Tage später.

Gut gemacht: Eine Seite, die Sie einer fremden Person geben könnten, und sie würde verstehen, was Sie bauen, für wen, und woran Sie den Erfolg messen.

Klein anfangen: Wählen Sie zuerst die einfachste nützliche Version des Problems. Bringen Sie diese eine Sache richtig gut zum Laufen und erweitern Sie sie später in den Iterationen (Schritt 6). Versuchen Sie nicht, alles auf einmal zu bauen — eine einfache Sache, die funktioniert, schlägt eine komplexe, die nur halb funktioniert.

Schritt 1 — Das Kontextpaket zusammenstellen

Was: Sammeln Sie in Dokumenten alles, woraus die KI bauen soll: das Ideendokument, das Ziel, Ihre Absichten, eine Funktionsliste, die Erfolgs- und Misserfolgsbeispiele, Ihre Werte und Rahmenbedingungen sowie Ihre Programmierstandards. Erstellen Sie daraus ein kurzes Designdokument.

Warum es wichtig ist: KI ist zuverlässig, wenn das Material vor ihr liegt, und unzuverlässig, wenn sie aus dem Gedächtnis abrufen muss. Ein gutes Kontextpaket ist zugleich Ihre stärkste Verteidigung gegen halluzinierte Ausgaben — Sie verankern die Arbeit in Fakten, die Sie geliefert haben.

Gut gemacht: Ein Ordner mit klaren Dokumenten. Nichts Wesentliches existiert nur in Ihrem Kopf oder in einem Chat-Verlauf, den Sie verlieren werden.

> **[Entwickler]** Nehmen Sie Schnittstellenerwartungen, Datenstrukturen und Nicht-Verhandelbares auf (Sicherheitsanforderungen, Lizenzen, Ziel-Laufzeitumgebung). Das wird die Quelle, gegen die die Spezifikation geschrieben wird.

Schritt 2 — Die Spezifikation schreiben

Was: Lassen Sie ein starkes Modell aus dem Kontextpaket eine schriftliche Spezifikation entwerfen. Dann lesen Sie sie, korrigieren sie und geben sie frei.

Warum es wichtig ist: Die Spezifikation ist der Vertrag. Fehler, die hier gefunden werden, sind billig; dieselben Fehler nach dem Bauen sind es nicht.

Gut gemacht: Eine Spezifikation, die Sie persönlich gelesen haben und der Sie wirklich zustimmen — nicht eine, die Sie überflogen haben.

> **[Entwickler]** Verwenden Sie dafür ein erstklassiges Reasoning-Modell (siehe „Das richtige Modell für die Aufgabe"). Die Spezifikation sollte präzise genug sein, um Mehrdeutigkeit auszuräumen, aber nicht so starr, dass sie gute Umsetzungsentscheidungen ausschließt.

Schritt 3 — Einen detaillierten Bauplan schreiben

Was: Erstellen Sie aus der Spezifikation einen detaillierten, schrittweisen Bauplan — detailliert genug, dass ein kleineres, günstigeres Modell ihn bis zur Fertigstellung ausführen könnte.

Warum es wichtig ist: Ein starker Plan erlaubt es Ihnen, die Hauptarbeit an wirtschaftliche Modelle zu delegieren, und hält den Build modular statt verworren.

Gut gemacht: Einzelne, geordnete Aufgaben mit klaren Schnittstellen dazwischen. Jemand (oder etwas) weniger Fähiges als der Planer könnte ihm folgen.

> **[Entwickler]** Bauen Sie modular, mit einem Kommunikationsbus zwischen den Komponenten als Standard, sofern es keinen guten Grund dagegen gibt. Definieren Sie die Nahtstellen, an denen sich das System natürlich trennt.

Schritt 4 — Mit Disziplin bauen

Was: Führen Sie den Plan aus. Bauen Sie antifragil und von der ersten Zeile an sicher. Arbeiten Sie testgetrieben, in Modulen. Führen Sie Tests fortlaufend während des Bauens aus und lassen Sie jeden Test lokal bestehen, bevor Sie pushen.

Warum es wichtig ist: Sicherheit und Widerstandsfähigkeit, die am Ende nachgerüstet werden, passen nie richtig. Antifragiles Design bedeutet, dass der Ausfall einer Komponente nicht das ganze System mitreißt.

Gut gemacht: Ein modularer Build, dessen Tests lokal grün sind, und bei dem der Ausfall eines Teils sich kontrolliert abschwächt, statt alles zusammenbrechen zu lassen.

> **[Entwickler]** Testgetriebene Entwicklung durchgängig. Antifragilität ist eine Design-Anforderung, kein Nice-to-have — isolieren Sie Fehlerdomänen. Kein Push bei Rot.

Schritt 5 — Unabhängige Prüfung & Härtung

Was: Lassen Sie **zwei getrennte** KI-Sitzungen die Arbeit adversarial kritisieren. Sie kritisieren nur — sie beheben nichts. Ihre Befunde gehen zurück an den ursprünglichen Ersteller zur Behebung. Führen Sie Penetrationstests mehrfach durch.

Warum es wichtig ist: Der Buchhalter prüft nicht seine eigenen Bücher. Ein Prüfer ohne eigenen Anteil an der Arbeit findet, wofür der Autor blind ist. Genau die Trennung ist der Punkt.

Gut gemacht: Die Kritik ist konkret und wird umgesetzt; Ausführender und Prüfer sind nie derselbe Agent; Penetrationstests wurden wiederholt, nicht nur einmal durchgeführt.

> **[Entwickler]** Frischer Kontext pro Prüfer — kein gemeinsames Gedächtnis mit dem Ersteller. Wiederholen Sie die Prüf-/Behebungsschleife, bis die Prüfer keine wesentlichen Probleme mehr finden.

Nicht verhandelbar: Lassen Sie alles durch eine CI/CD-Pipeline laufen, jedes Mal — automatisch gebaut und getestet bei jeder Änderung. Alle Tests bestehen, immer; ein roter Build wird nie

gemergt und nie „zum späteren Beheben” ausgeliefert. Und Penetrationstests sind ein absolutes Muss, kein Schritt, den man bei Zeitdruck überspringt — ein ungetestetes System ist ein nicht vertrauenswürdiges System.

Schritt 6 — Menschliches Testen, dann iterieren

Was: Der erste menschliche Test findet statt, wenn das System vollständig läuft. Von dort aus entwickeln Sie iterativ weiter.

Warum es wichtig ist: Ein halbfertiges System zu testen verschwendet die Aufmerksamkeit aller an Problemen, die der Build ohnehin gelöst hätte. Echtes Feedback gehört an eine echte, laufende Sache.

Gut gemacht: Ein laufendes System vor einem Menschen und eine kurze Liste der nächsten Verbesserungen — kein Gerangel, es überhaupt zum Starten zu bringen.

Zwei Regeln, die durch jeden Schritt laufen

Das richtige Modell für die Aufgabe. Geben Sie das schwierigste Denken — Umreißen, Spezifikation, Plan, Kritik — dem besten verfügbaren Modell. Geben Sie die Routineausführung dem günstigsten Modell, das die Aufgabe gut erledigt. Werkzeug und Aufgabe aufeinander abzustimmen ist zugleich zuverlässiger und günstiger.

> **[Entwickler]** Eine Arbeitsklassifizierung, über die Zeit angepasst (Namen und Versionen ändern sich): Spitzenklasse für schwieriges Reasoning; leistungsfähige Mittelklasse und wirtschaftliche Modelle für die Routineausführung. Benchmarken Sie regelmäßig neu — diese Liste veraltet schnell.

Trennen Sie den Ausführenden vom Prüfer. Derselbe Interessenkonflikt, der Buchhalter davon abhält, sich selbst zu prüfen, gilt für KI. Wenn Sie entscheiden, wie Arbeit aufgeteilt und zugewiesen wird, stellen Sie vier Fragen:

1. **Größe** — wie groß ist die Aufgabe in Tokens für eine Sitzung?
2. **Unabhängigkeit** — lässt sie sich aufteilen, ohne an Wirksamkeit zu verlieren?
3. **Trennung der Zuständigkeiten** — entsteht ein Interessenkonflikt, wenn ein Agent beides tut?
4. **Prüfbarkeit** — wenn Prüfen viel billiger ist als Tun, setzen Sie mehrere Prüf-Agenten ein.

Vor dem Release — der 90-Sekunden-Check

> 1. Habe ich das eigentliche Problem umrissen oder nur das erste, das mir einfiel?
> 2. Ist die Arbeit in einem Kontextpaket verankert oder im Gedächtnis des Modells?
> 3. Hat ein unabhängiger Agent sie geprüft — nicht der, der sie gebaut hat?
> 4. Bestehen alle Tests lokal, und hat es wiederholte Penetrationstests überstanden?
> 5. Hat ein Mensch das vollständig laufende System benutzt?

Wenn jede Antwort Ja lautet, sind Sie in deutlich besserer Verfassung als die meisten.

Dies ist ein allgemeiner Überblicksleitfaden zum Bauprozess — er bleibt auf der Ebene von Prinzip und Reihenfolge und kann die Besonderheiten eines einzelnen Projekts nicht abdecken. Für detaillierte, auf Ihre Situation zugeschnittene Beratung nehmen Sie Kontakt auf.

Über Rehm KI Consulting

Wir helfen Unternehmen, richtig mit KI zu bauen — mit der ingenieurmäßigen Disziplin, die ein verlässliches System von einer beeindruckenden Demo unterscheidet. Klare Antworten darauf, wo KI passt und wo nicht. Kein Hype.

Rehm KI Consulting · Christopher Rehm · Bayern, Deutschland · contact@christopherrehm.de