KI FÜR ENTWICKLER · TEIL 3

Arbeiten mit LLM-APIs

In Teil 2 schauten wir in das Modell.

Jetzt verbinden wir es mit einer Anwendung.

Anfrage senden

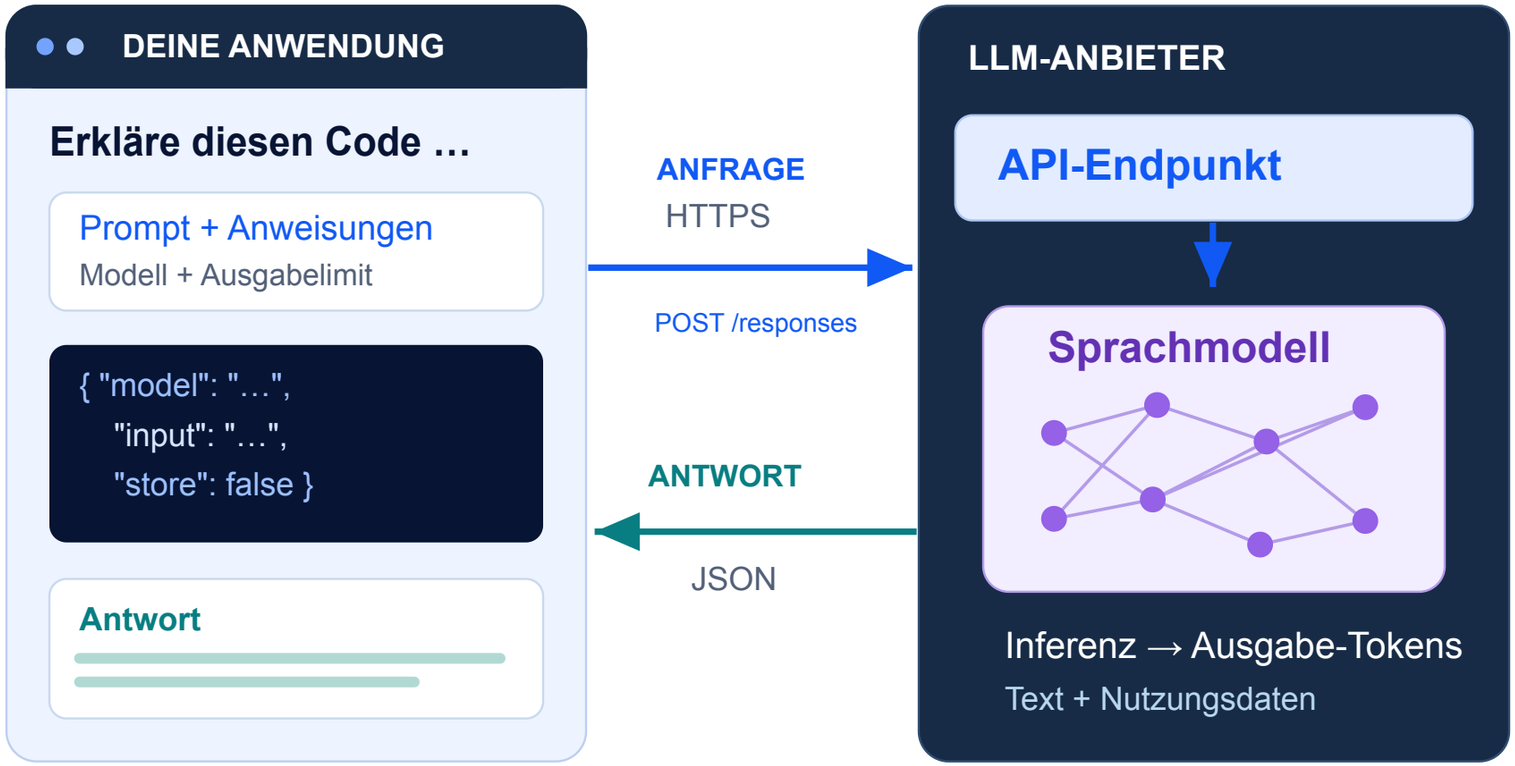
HTTP, Authentifizierung und JSON.

Kontext gestalten

Anweisungen und Gesprächsverlauf.

Zuverlässig entwickeln

Validieren, messen und Fehler behandeln.



UNSER WEG Anfrage → Antwort → Kontext → zuverlässige Anwendung

OpenAI-Doku · Textgenerierung

Rückblick: Vom Modell zur Anwendung

Teil 2 öffnete die Blackbox. Vier Ideen erklären den Ablauf hinter einer LLM-API.

1

Tokens → Vektoren

bank →



Token-IDs werden erlernten Embedding-Vektoren zugeordnet.

2

Schichten bilden Kontext

Kausale Attention + Position

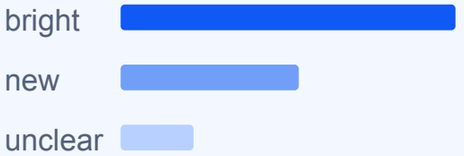
MLP + Residualpfade

Über Decoder-Blöcke wiederholen.

Jede Position nutzt ihre Vergangenheit.

3

Nächstes Token vorhersagen



Vokabular-Scores → Wahrscheinlichkeiten.

Schematische Verteilung.

4

Anhängen und wiederholen

The future of AI is

bright

Token wählen; erneut vorhersagen.

Bei Inferenz bleiben Gewichte fest.

Kontext ist begrenzt.
Deine App wählt Verlauf und Belege für die Anfrage.

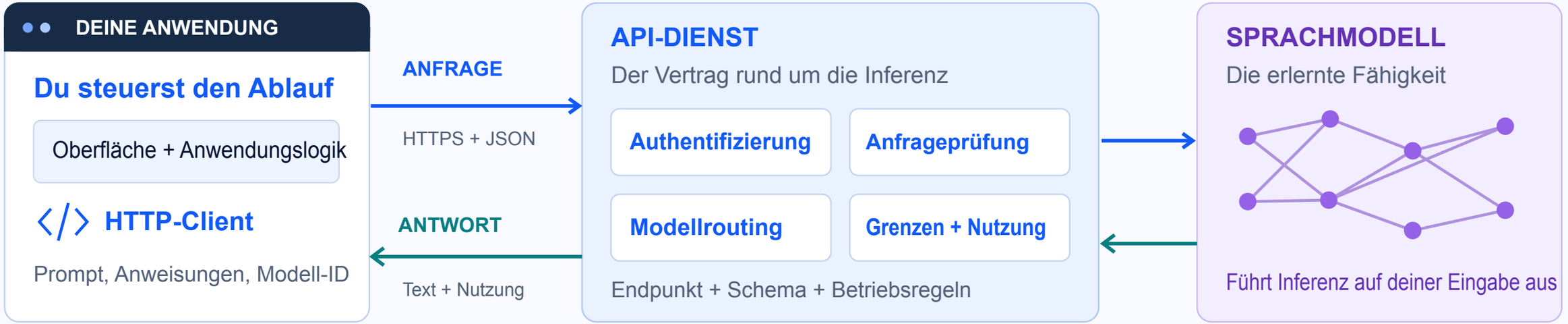
Flüssig heißt nicht geprüft.
Wichtige Aussagen belegen und Ergebnisse validieren.

TEIL 3 **Deine Anwendung → LLM-API → dieser Mechanismus → nützliche, geprüfte Ausgabe**

Rückblick auf Teil 2 · LLM-Grundlagen II: Im Modell

Was ist eine LLM-API?

Eine programmierbare Schnittstelle, über die deine Anwendung ein Sprachmodell im Netzwerk nutzt.



API = Schnittstelle + Dienst
So greifst du auf die Fähigkeit zu und nutzt sie.

Modell = erlernte Parameter + Inferenz
Erzeugt die Ausgabe aus deinem Kontext.

Ein Chatprodukt bietet eine fertige Oberfläche. Mit einer API baust du deine eigene.

OpenAI-Doku · Textgenerierung

Der Aufbau einer API-Anfrage

Eine HTTPS-Anfrage verbindet Ziel, Zugangsdaten und einen strukturierten JSON-Body.

1 **Ziel**
POST startet eine Generierungsanfrage.
Nutze den HTTPS-Endpunkt des Anbieters.

2 **Authentifizierung**
Der Header enthält deinen API-Schlüssel.
Zur Laufzeit übergeben; nie fest einbauen.

3 **Modell**
Wähle die benötigte Fähigkeit.
Dein Schlüssel braucht Zugriff auf das Modell.

4 **Eingabe**
Sende den Nutzer-Prompt oder Eingabeelemente.
Ergänze bei Bedarf Anweisungen und Kontext.

5 **Steuerung**
Ausgabe begrenzen; Speicherung wählen.
Unterstützte Optionen hängen vom Modell ab.

HTTP-ANFRAGE · OPENAI RESPONSES API

```
1 POST https://api.openai.com/v1/responses
2 Authorization: Bearer RUNTIME_KEY
  Content-Type: application/json
  {
3   "model": "gpt-5.4-mini",
4   "input": "Explain an API in 3 sentences.",
5   "max_output_tokens": 400,
    "store": false
  }
```

JETZT ABSENDEN **Auf Folie 5 temporären Schlüssel und Prompt eingeben – dann die Antwort untersuchen.**

OpenAI-Doku · Textgenerierung

Dein erster LLM-API-Aufruf

LIVE-DEMO

01 MIT OPENAI VERBINDEN

API-Schlüssel

Temporären API-Schlüssel eingeben

Modell

gpt-5.4-mini

02 DEINE ANFRAGE FORMULIEREN

Prompt

Erkläre in 3 Sätzen, wie ein LLM funktioniert.

Senden →

Leeren

Browser-CORS prüfen

03 · OpenAI-Antwort

POST /v1/responses → JSON → Text

```
Hier erscheint die Antwort des Modells.
```

Eingabe-Tokens: – · Ausgabe-Tokens: – · Zeit: –

Direkt vom Browser zu OpenAI. Kein Proxy. CORS wurde noch nicht geprüft.

> Der Schlüssel bleibt nur zur Laufzeit auf dieser Seite; Leeren entfernt ihn und bricht die Anfrage ab. Browser-Tools und Seitenskripte haben Zugriff. Temporären Schlüssel nutzen und danach widerrufen. Der Prompt geht an OpenAI; store:false schließt eine Datenaufbewahrung nicht aus.

Die API-Antwort verstehen

Die Antwort ist ein strukturiertes Ergebnis – erzeugter Text ist nur ein Teil davon.

ANTWORT-BODY · VEREINFACHTES JSON

```
{
  "status": "completed",                                    1
  "output": [{
    "type": "message",
    "content": [{
      "type": "output_text",
      "text": "An API lets apps communicate."               2
    }]
  }],
  "usage": {
    "input_tokens": 24,
    "output_tokens": 38                                     3
  }
}
```

JSON und Tokenzahlen sind Beispiele, keine gemessene Demo-Antwort.

1 Ergebnisstatus prüfen

Vor der Nutzung den Status lesen.

Unvollständige Ausgabe, Ablehnungen und Fehler behandeln.

2 Erzeugten Text extrahieren

`output[] → message → content[] → output_text`

Passende Textteile sammeln und als Text darstellen.

Nicht annehmen, dass output[0] immer eine Message ist.

3 Tatsächlichen Verbrauch lesen

Eingabe: 24 Tokens Ausgabe: 38 Tokens

Die Laufzeit misst der Browser separat.

ROHES REST ≠ SDK-HELFER Der SDK-Helfer output_text sammelt Text; fetch liefert typisierte Ausgabeelemente.

OpenAI-Doku · Textgenerierung

Anweisungen und Gesprächskontext

Gib dem Modell Verhaltensregeln und die nötigen Informationen für diese Anfrage.

01 · ANWEISUNGEN

Verhalten der Anwendung festlegen.

Aufgabe, Zielgruppe, Grenzen und Format.

Hinweise klar und konsistent halten.

02 · EINGABE + KONTEXT

Frage und Belege mitgeben.

Aktuelle Nutzernachricht

Relevante frühere Nutzer-/Assistentenbeiträge

Bei Bedarf Dokumente oder Tool-Ergebnisse

Deine App wählt aus, was sie mitgibt.

RESPONSES API · AUSZUG AUS DEM ANFRAGE-BODY

```
{
  "instructions": "Explain APIs clearly. Be concise.",

  "input": [
    { "role": "user",
      "content": "What is an API?" },
    { "role": "assistant",
      "content": "An API lets applications communicate." },
    { "role": "user",
      "content": "Now give me a GET request example." }
  ]
}
```

GRENZEN BEACHTEN

Abgerufener Text ist nicht vertrauenswürdig. Keine Geheimnisse in Prompts.
Anweisungen lenken das Verhalten; wichtige Ausgaben in der Anwendung validieren.

OpenAI-Doku · Textgenerierung

Modellverhalten steuern

Nutze die vom Modell unterstützten Regler und miss, ob das Ergebnis deine Anforderungen erfüllt.

1

Fähigkeit auswählen

MODELL
Modell passend zur Aufgabe

Modellstärken mit der Aufgabe abgleichen.

Qualität, Latenz und Kosten vergleichen.

An repräsentativen Beispielen evaluieren.

2

Ausgabe begrenzen

max_output_tokens



Explizite Obergrenze setzen.

Bei Unterstützung ein Schema verlangen.

Zurückgegebene Struktur validieren.

Ein Tokenlimit kann die Antwort abschneiden.

3

Temperatur

API-Parameter: temperature

Niedrig · z. B. 0,2

Bevorzugt wahrscheinliche Tokens; weniger Variation.

Hoch · z. B. 0,8

Variablere Token-Auswahl.

Nur bei Unterstützung durch Modell
und gewählte Konfiguration.

VARIATION EINPLANEN **Niedrige Temperatur garantiert keine identischen Antworten.**

AUSZUG AUS DEM API-BODY

```
{ "temperature": 0.2 }
```

Kein Regler für Wahrheit oder Qualität.
Unterstützung prüfen, sonst weglassen. Folie 5 setzt den Parameter noch nicht.

OpenAI-Doku · Textgenerierung

Gespräche über mehrere Runden

Für eine Nachfrage sendet deine App den bisherigen Austausch zusammen mit der neuen Frage.

DAS GESPRÄCH

NUTZER · ERSTE FRAGE
What is an API?

ASSISTENT · ERSTE ANTWORT
An API lets applications communicate.

NUTZER · NACHFRAGE
Now show me a GET request example.

Der nächste Aufruf erhält alle drei Nachrichten.

Kein automatisches Gedächtnis vorausgesetzt.

ZWEITE ANFRAGE · BODY-AUSZUG

```
{
  "instructions": "Explain APIs clearly. Be concise.",
  "input": [
    {
      "role": "user",
      "content": "What is an API?"
    },
    {
      "role": "assistant",
      "content": "An API lets applications communicate."
    },
    {
      "role": "user",
      "content": "Now show me a GET request example."
    }
  ]
}
```

WEITERE OPTION

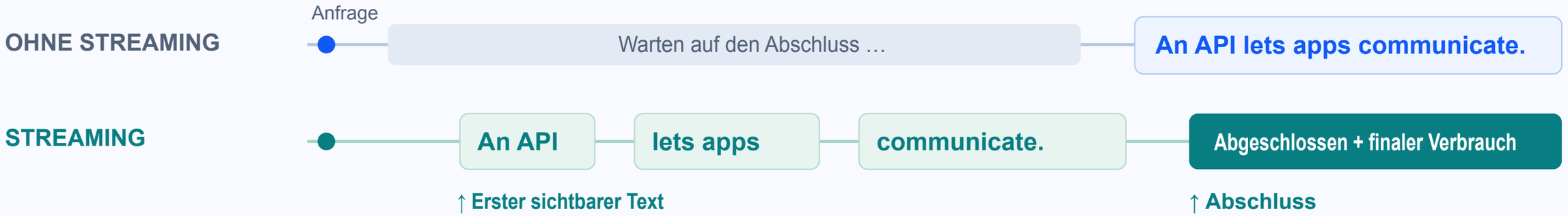
Unterstützten Anbieterzustand nutzen, z. B. previous_response_id; nötige Anweisungen erneut senden.
Aufbewahrung und Kontextwachstum prüfen. Folie 5 sendet weiterhin unabhängige Anfragen.

OpenAI-Doku · Gesprächszustand

Antworten streamen

Text beim Eintreffen anzeigen, statt auf den Abschluss der gesamten Antwort zu warten.

GLEICHE AUFGABE · ZWEI ARTEN DER AUSGABE



Zeiten sind schematisch. Früher sichtbarer Text garantiert keine kürzere gesamte Generierungszeit.

01 · ANFRAGE

{ "stream": true }

Streaming im JSON-Body aktivieren.

02 · LESEN + DARSTELLEN

SSE-Events lesen; Text-Deltas anhängen.

Netzwerk-Chunks ≠ vollständige Events.

03 · SICHER ABSCHLIESSEN

Abschluss und Fehler behandeln.

Abbruch ermöglichen; als Text darstellen.

UNSERE LIVE-DEMO Folie 5 nutzt kein Streaming. Dafür wäre zusätzlicher Code zur Eventbehandlung nötig.

OpenAI-Doku · Streaming-Antworten

Tokens, Latenz und Kosten

Miss die ganze Anfrage: Verbrauch, Dauer und Kosten.

01 · TOKENS

Eingabe und Ausgabe zählen

EINGABE

Anweisungen | Verlauf | Prompt

AUSGABE

Erzeugte Tokens

Nutzungsdaten aus der API lesen.

Tokens sind nicht dasselbe wie Wörter.

02 · LATENZ

Die verstrichene Zeit verfolgen



Netzwerk → Eingabeverarbeitung → Modellberechnung → Ausgabeübertragung

Streaming: erster Text ≠ Abschluss.

Demo-Zeit: vom Browser gemessene Gesamtdauer.

03 · KOSTEN

Andere Modelle, andere Preise

Gleiche Tokenzahl ≠ gleiche Kosten.

Jedes Modell hat eigene Ein-/Ausgabepreise.

Ausgabe ist meist je Token teurer.

Lange Eingabe kann insgesamt mehr kosten.

Gecachte Eingabe hat eigene Preise.

Kosten = Tokenzahl × geltender Preis

Kategorien summieren; aktuelle Modellpreise nutzen.

GESAMTKOSTEN ZÄHLEN Eingabe- plus Ausgabekosten; bei Preisen je Million die Tokenzahlen durch 1 Mio. teilen.

JE AUFGABE VERGLEICHEN **Qualität + Latenz + Gesamtkosten**

Tokenzahlen variieren je Modell. Cache und Reasoning beeinflussen die Abrechnung; Demo-Zeiten sind kein Preisangebot.

OpenAI-Doku · Aktuelle Modellpreise

Fehler und Grenzen behandeln

Zuerst die Fehlerart bestimmen. Wiederholen hilft nur bei vorübergehenden Ursachen.

SIGNAL	WAS PRÜFEN?	AKTION DER ANWENDUNG
400 / 404	Anfrage, Parameter, Endpunkt oder Modellzugriff	Anfrage oder Konfiguration korrigieren.
401 / 403	Schlüssel, Projektrechte und Zugriff	Zugangsdaten / Rechte korrigieren; nicht blind wiederholen.
429	Ratenlimit oder Quote erschöpft? Fehler lesen.	Ratenlimit: Backoff. Quote: Konto prüfen.
5xx	Vorübergehender Fehler beim Anbieter	Begrenzt mit Backoff + Jitter wiederholen.
TIMEOUT	Die Anfrage könnte trotzdem abgeschlossen sein	Lokal abbrechen; Ergebnis vor Wiederholung prüfen.

BROWSER-FETCH FEHLGESCHLAGEN?

Netzwerk, CORS oder Browserregeln können den Aufruf sperren.
JavaScript kann diese Ursachen nicht immer unterscheiden.

WIEDERHOLUNGSREGEL · SCHEMATISCH

Warten → länger warten → nach begrenzter Zahl stoppen
Jitter ergänzen; Anbieterhinweise und Fristen beachten.

SICHER SCHEITERN — **Nützliche Meldungen zeigen. Nie API-Schlüssel loggen. Anfrage-IDs zur Diagnose behalten.**

OpenAI-Doku · Fehlercodes

Das passende Modell wählen

Nach gemessenen Ergebnissen für deine Aufgabe wählen, nicht allein nach dem Modellnamen.

01 · ANFORDERUNGEN DEFINIEREN

Was muss das Modell leisten?

- ☐ Qualität und akzeptable Fehlerrate
- ☐ Ziel für die Antwortzeit
- ☐ Budget je abgeschlossener Aufgabe
- ☐ Ausgabeformat und Tool-Unterstützung
- ☐ Datenschutz und Betriebsanforderungen

02 · KANDIDATEN AN GLEICHEN AUFGABEN TESTEN

Normale Eingaben, schwierige Fälle und erwartete Fehler einbeziehen.

Kandidat	Qualität	Latenz	Kosten / Aufgabe
Modell A	Messen	Messen	Messen
Modell B	Messen	Messen	Messen
Modell C	Messen	Messen	Messen

Nur eine Vorlage – keine erfundenen Scores oder allgemeine Modellrangfolge.

03 · AUSWÄHLEN

Erst die Qualitätsschwelle erfüllen; dann Gesamtkosten und Antwortzeit optimieren.

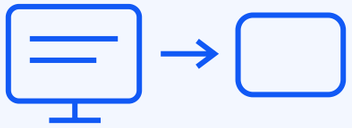
VOR DEM EINSATZ PRÜFEN — **Modellzugriff + Parameter + Grenzen** → bei Änderungen neu evaluieren

OpenAI-Doku · Modellauswahl

Verschiedene API-Anbieter

Die Anfrage läuft über HTTP – Dienstvertrag und Betriebsmodell können sich jedoch unterscheiden.

01 · DIREKT GEHOSTETE API



Den Anbieter direkt aufrufen.

Beispiele: OpenAI, Anthropic, Google

Der Anbieter betreibt das Modell.

Seine Zugangsdaten, Schemas und Abrechnung nutzen.

02 · CLOUD-PLATTFORM



Eine Cloud-Zugriffsschicht nutzen.

Microsoft Azure · Amazon Bedrock

Cloud-Identität und Bereitstellungsoptionen.

Modelle und Endpunkte variieren.

03 · SELBST GEHOSTET



Den Modellserver selbst betreiben.

Eigene Infrastruktur + Modellserver

Hardware, Skalierung und Monitoring selbst verantworten.

Modelllizenz und Fähigkeiten prüfen.

VOR EINEM WECHSEL PRÜFEN

- **Authentifizierung + Schemas**
- **Streaming + Tools + Ausgabeformate**
- **Aufbewahrung + Nutzungsabrechnung**

GRUNDIDEE **Eine ähnliche HTTP-Schnittstelle garantiert kein identisches Verhalten. Integration testen.**

Eine einfache LLM-Anwendung bauen

Beispiel: Ein Code-Erklärer macht aus einem eingesendeten Ausschnitt eine klare Erklärung.

01 · BENUTZEROBERFLÄCHE

```
const total = items.reduce(
(sum, x) => sum + x, 0);
```

Codeausschnitt entgegennehmen.

Ladezustand, Antwort oder Fehler zeigen.

Generierten Text sicher darstellen.

02 · DEIN BACKEND

Eingabe prüfen + Anweisungen ergänzen

Modell wählen + Ausgabe begrenzen

Anbieter mit serverseitigem Schlüssel aufrufen

Antwort vor der Rückgabe prüfen

Ablauf und Betriebsregeln selbst steuern.

03 · LLM-ANBIETER

Anweisungen + Codeausschnitt
→ Inferenz → Erklärung

Erklärung erzeugen.

Ausgabe + Nutzungsdaten zurückgeben.

Deine App validiert wichtige Ergebnisse.

VOR DEM PRODUKTIVBETRIEB

- **Zugriff + Eingabegrenzen**
- **Timeouts + Budgets**
- **Logs ohne Geheimnisse**
- **Qualitätsevaluationen**

LEHR-DEMO Folie 5 nutzt einen temporären Browser-Schlüssel. Im Produktivbetrieb bleiben Zugangsdaten auf dem Server.

DEINE ÜBUNG **Baue den Code-Erklärer; teste nützliche Antworten, ungültige Eingaben und Anbieterfehler.**

KI FÜR ENTWICKLER · TEIL 3

Kernaussagen & Ausblick

Eine Anfrage ist einfach. Eine zuverlässige Anwendung braucht Entwicklungsarbeit.

01 Den Vertrag verstehen

Endpunkt, Authentifizierung und JSON.

Typisierte Ausgabe lesen; Abschluss behandeln.

02 Kontext gestalten

Klare Anweisungen und relevanten Verlauf mitgeben.

Abgerufene Inhalte als nicht vertrauenswürdige Daten behandeln.

03 Das System messen

Qualität, Latenz und Kosten je Aufgabe vergleichen.

Modellpreise und unterstützte Regler unterscheiden sich.

04 Fehler einplanen

Ausgabe validieren und Zugangsdaten schützen.

Grenzen, Fehler und unklare Ergebnisse behandeln.

WEITER · TEIL 4

Prompting & strukturierte Ausgabe

System-Prompts · Few-Shot-Prompting · JSON-Ausgaben · Schemas · zuverlässiges Prompt-Design

ZUM MITNEHMEN

Mit einer nützlichen Aufgabe starten. Erfolg definieren. Fehler ebenso testen wie den Normalfall.