Wie LLMs Fragen beantworten

Von der Frage zur Antwort: Transformer, Wahrscheinlichkeiten, Token-Generierung.

Grundlagen · Kapitel 9 von 50 · ca. 20 Min. · 100 XP

Kapitel online

Einfach gesagt: ChatGPT schlägt deine Antwort nicht nach, sondern errät sie Wort für Wort — es wählt einfach das wahrscheinlichste nächste Wortstück.

Stell dir vor: Du fragst ChatGPT "Was ist KI?" und bekommst eine perfekte Antwort. Aber **wie** generiert das Modell Wort für Wort die Antwort? Das zeige ich dir jetzt!

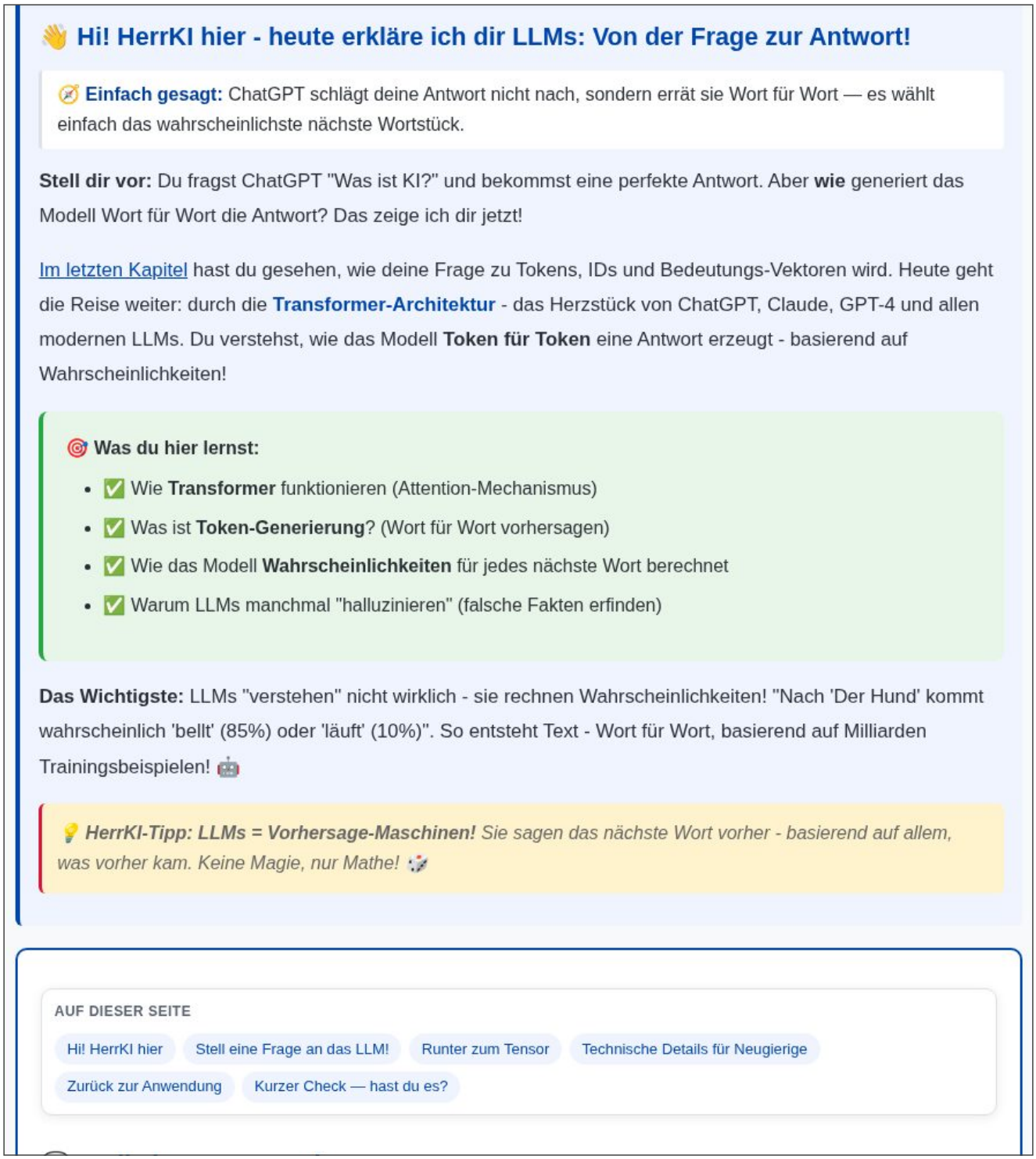
Im letzten Kapitel hast du gesehen, wie deine Frage zu Tokens, IDs und Bedeutungs-Vektoren wird. Heute geht die Reise weiter: durch die **Transformer-Architektur** - das Herzstück von ChatGPT, Claude, GPT-4 und allen modernen LLMs. Du verstehst, wie das Modell **Token für Token** eine Antwort erzeugt - basierend auf Wahrscheinlichkeiten!

Das Wichtigste: LLMs "verstehen" nicht wirklich - sie rechnen Wahrscheinlichkeiten! "Nach 'Der Hund' kommt wahrscheinlich 'bellt' (85%) oder 'läuft' (10%)". So entsteht Text - Wort für Wort, basierend auf Milliarden Trainingsbeispielen!

HerrKI-Tipp: LLMs = Vorhersage-Maschinen! Sie sagen das nächste Wort vorher - basierend auf allem, was vorher kam. Keine Magie, nur Mathe!

Am Gerät ausprobieren: herrki.de/050_wie_llms_antworten#abschnitt-1

Direkt zum Modul



Interaktives Modul – am Gerät ausprobieren

Was du hier lernst

- Wie **Transformer** funktionieren (Attention-Mechanismus)
- Was ist **Token-Generierung**? (Wort für Wort vorhersagen)

- Wie das Modell **Wahrscheinlichkeiten** für jedes nächste Wort berechnet
- Warum LLMs manchmal "halluzinieren" (falsche Fakten erfinden)

Für den Unterricht

Einstieg (4 Min): Kapitel gemeinsam lesen, Alltagsbeispiel aufgreifen.

Erkunden am Gerät (12 Min): die interaktiven Module auf herrki.de/050_wie_llms_antworten ausprobieren.

Sichern mit Quiz (4 Min): „Kurzer Check" besprechen, „Das nimmst du mit" festhalten.

Stell eine Frage an das LLM!

Lass uns eine einfache Frage stellen:

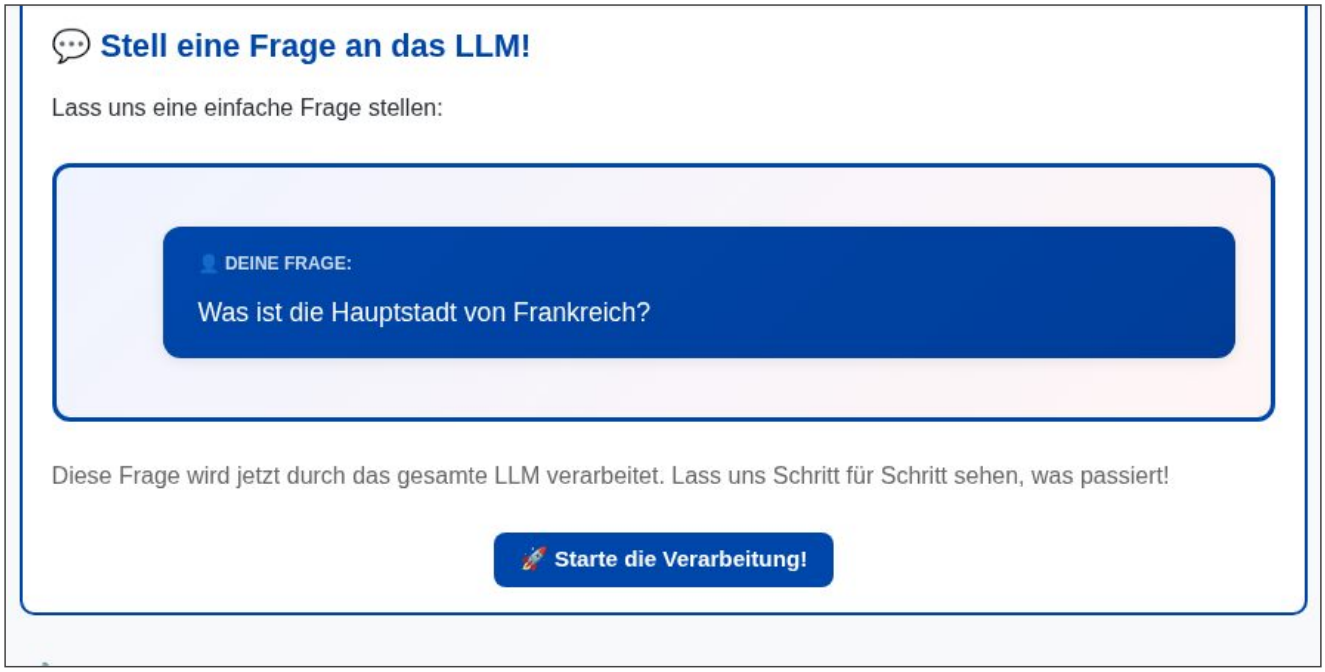
DEINE FRAGE:

Was ist die Hauptstadt von Frankreich?

Diese Frage wird jetzt durch das gesamte LLM verarbeitet. Lass uns Schritt für Schritt sehen, was passiert!

Am Gerät ausprobieren: herrki.de/050_wie_llms_antworten#abschnitt-2

Direkt zum Modul



Interaktives Modul – am Gerät ausprobieren

Runter zum Tensor: Sechs Schritte von der Frage zur Antwort

Jetzt gehen wir unter die Haube. Zwischen deiner Frage und dem Wort „Paris“ liegen genau sechs Schritte — und in jedem einzelnen ist deine Frage nichts anderes als ein Zahlenpaket, das seine Form ändert. Klick oben auf „Starte die Verarbeitung“, dann klappen die Schritte hier auf.

Schritt 1 von 6

1 Tokenization: Zerlege den Text

Der erste Schritt: Die Frage wird in Tokens zerlegt.

↓

Anzahl Tokens: 0

Schritt 2 von 6

2 Token-IDs: Wandle in Zahlen um

Jedes Token bekommt seine eindeutige ID aus dem Vokabular.

↓

Tensor Shape: [0]

Schritt 3 von 6

3 Embeddings: Bedeutung kodieren

Jede Token-ID wird zu einem 768-dimensionalen Vektor.

Was → [0.23, -0.45, 0.67, ...] (768 Zahlen)
ist → [-0.12, 0.89, -0.34, ...] (768 Zahlen)
die → [0.56, 0.23, -0.78, ...] (768 Zahlen)
...

Embedding Tensor Shape: [0, 768]

[Anzahl_Tokens, Embedding_Dimension]

Schritt 4 von 6

4 Transformer: Hier wird der Kontext verrechnet

Der Transformer verarbeitet alle Embeddings gleichzeitig und "versteht" den Kontext. Und auch wenn es gleich nach Magie aussieht: Es sind ausschließlich **Matrix-Multiplikationen auf Tensoren** - die Shape bleibt die ganze Zeit [Anzahl_Tokens, 768]. Wie das „Schauen auf andere Wörter" (Attention) im Detail funktioniert, bekommst du in „Worauf schaut die KI gerade?" zum Anfassen.

Was macht der Transformer?

Self-Attention: Welche Wörter sind wichtig?

Der Transformer schaut sich jedes Wort im Kontext aller anderen Wörter an:

Was → schaut auf → "ist", "Hauptstadt", "Frankreich"
Hauptstadt → schaut auf → "Was", "von", "Frankreich"
Frankreich → schaut auf → "Was", "Hauptstadt", "von"

→ **Ergebnis:** Der Transformer "versteht", dass die Frage nach der Hauptstadt VON Frankreich fragt. "Frankreich" und "Hauptstadt" sind besonders wichtig!

Multiple Layers (z.B. 12-96 Schichten):

- **Layer 1:** Erkenne einzelne Wörter und einfache Beziehungen
- **Layer 2-5:** Verstehe Satzbau und Grammatik
- **Layer 6-10:** Verstehe semantische Bedeutung
- **Layer 11+:** Verstehe komplexe Konzepte und Wissen

→ Am Ende hat der Transformer eine "Repräsentation" der Frage,
die alle Bedeutung und Kontext enthält!

Output Shape nach Transformer: [0, 768]

Immer noch [Anzahl_Tokens, 768], aber jetzt mit viel mehr "Verständnis"!

Schritt 5 von 6

5 Output Layer: Welches Token kommt als Nächstes?

Jetzt wird's spannend! Der Output-Layer muss entscheiden, welches Wort als Nächstes kommt.

Wie entscheidet das LLM?

Analogie: Wie in einer Schulklasse!

Stell dir vor, die Lehrerin fragt: **"Was ist die Hauptstadt von Frankreich?"**
In der Klasse sitzen 50.000 Schüler (alle möglichen Tokens), und jeder hebt die Hand mit unterschiedlicher Energie:

"Paris"
Meldet sich SEHR enthusiastisch!

"Die"
Meldet sich zaghaft...

"Computer"
Schläft fast...

Die Lehrerin (das LLM) zählt, wie energisch sich jeder Schüler meldet:
→ Das ist die **Wahrscheinlichkeitsverteilung**!

Was berechnet der Output-Layer genau?

Der Transformer hat die Frage "verstanden" und gibt einen Vektor aus: [768]
Der Output-Layer wandelt diesen Vektor in **50.000 Zahlen** um - eine für jedes mögliche Wort!

Schritt für Schritt:

Schritt 1: Matrix-Multiplikation

Der Transformer-Output [768] wird mit einer riesigen Matrix multipliziert
`[768] × [768, 50000] = [50000]`

Schritt 2: "Logits" (Roh-Scores)

Jetzt haben wir 50.000 Zahlen, aber die können beliebig groß/klein sein:

```
Paris: 15.3, Die: 2.1, Computer: -8.7, ...
```

Schritt 3: Softmax (→ Wahrscheinlichkeiten)

Die **Softmax-Funktion** wandelt die Roh-Scores in Wahrscheinlichkeiten um (0% bis 100%):

```
Paris: 92.5%, Die: 3.2%, Computer: 0.0001%, ...
```

Wichtig: Softmax sorgt dafür, dass alle Wahrscheinlichkeiten zusammen genau 100% ergeben!
Wenn ein Wort wahrscheinlicher wird, werden andere automatisch unwahrscheinlicher.

Die Wahrscheinlichkeitsverteilung im Detail:

Das LLM hat für **alle 50.000 Tokens** Wahrscheinlichkeiten berechnet. Hier sind die Top 5:

Top 5 wahrscheinlichste Tokens:

"Paris" 92.5%

92.5%

→ Extrem wahrscheinlich! Das LLM ist sich fast sicher.

"Die" 3.2%

3.2%

→ Grammatikalisch möglich, aber nicht die beste Wahl hier.

"Lyon" 1.8%

1.8%

→ Ist auch eine Stadt in Frankreich, aber nicht die Hauptstadt!

"Marseille" 0.9%

0.9%

→ Auch eine französische Stadt, aber sehr unwahrscheinlich.

"Frankreich" 0.4%

0.4%

→ Grammatikalisch falsch hier, daher sehr unwahrscheinlich.

...und 49.995 weitere Tokens mit sehr kleinen Wahrscheinlichkeiten (z.B. "Computer": 0.0001%, "laufen": 0.00001%, ...)

Alle zusammen = 100.0% ✓

Entscheidung: Das LLM wählt "Paris"!

Mit **92.5% Wahrscheinlichkeit** ist "Paris" das richtige nächste Token.
Das ist wie wenn sich ein Schüler mit voller Überzeugung meldet - die Lehrerin nimmt ihn natürlich dran!

Warum ist Paris so viel wahrscheinlicher?

Der Transformer hat während des Trainings **Tausende von Texten** über Frankreich und Paris gelesen:

- "Die Hauptstadt von Frankreich ist Paris."
- "Paris ist die größte Stadt Frankreichs."
- "Der Eiffelturm steht in Paris, der Hauptstadt."
- ...Tausende weitere Beispiele...

→ Das LLM hat gelernt: Wenn die Frage nach der Hauptstadt Frankreichs ist, ist die Antwort fast immer "Paris"!

Output Shape: [50257]

Eine Wahrscheinlichkeit für jedes Token im Vokabular (z.B. 50.257 bei GPT-2)!

Schritt 6 von 6

6 Token-für-Token Generierung

Das LLM generiert die Antwort Token für Token - wie beim Tippen!

Der Generierungs-Loop

Am Gerät ausprobieren: herrki.de/050_wie_llms_antworten#abschnitt-3

Direkt zum Modul

🧩 Technische Details für Neugierige

⚡ Warum ist das so schnell?

LLMs verarbeiten alle Tokens **parallel** (gleichzeitig) im Transformer! Nur die Generation ist sequentiell (Token für Token).

- **Input-Verarbeitung:** Alle Tokens auf einmal → sehr schnell!
- **Output-Generierung:** Ein Token nach dem anderen → langsamer
- **GPUs/TPUs:** Spezielle Hardware für parallele Berechnungen

🎲 Temperature & Sampling

Das LLM wählt nicht immer das wahrscheinlichste Token! Es gibt verschiedene Strategien:

- **Greedy Decoding:** Wähle immer das wahrscheinlichste Token → langweilig, aber präzise
- **Temperature Sampling:** Höhere Temperatur = mehr Kreativität/Zufall
- **Top-k Sampling:** Wähle zufällig aus den Top k wahrscheinlichsten Tokens
- **Nucleus (Top-p) Sampling:** Wähle aus Tokens, die zusammen p% Wahrscheinlichkeit haben

🔢 Größenordnungen (GPT-3.5 / GPT-4):

- **Parameter:** 175 Milliarden (GPT-3) bis über 1 Billion (GPT-4)
- **Layers:** 96+ Transformer-Layer
- **Embedding-Dimension:** 12.288 (statt 768)
- **Kontext-Länge:** 8.000 - 128.000 Tokens
- **Inference-Zeit:** ~50-100ms pro Token

Interaktives Modul – am Gerät ausprobieren

Vollständige Antwort:

Die Antwort erscheint hier...

Wichtig zu verstehen:

- Das LLM generiert **EIN Token nach dem anderen**
- Jedes neue Token wird zur Eingabe hinzugefügt
- Der gesamte Prozess (Tokenization → Embeddings → Transformer → Output) wird für jedes Token wiederholt!
- Das LLM stoppt, wenn es ein spezielles `[END]` Token generiert

Technische Details für Neugierige

Warum ist das so schnell?

LLMs verarbeiten alle Tokens **parallel** (gleichzeitig) im Transformer! Nur die Generation ist sequentiell (Token für Token).

- **Input-Verarbeitung:** Alle Tokens auf einmal → sehr schnell!
- **Output-Generierung:** Ein Token nach dem anderen → langsamer
- **GPUs/TPUs:** Spezielle Hardware für parallele Berechnungen

Temperature & Sampling

Das LLM wählt nicht immer das wahrscheinlichste Token! Es gibt verschiedene Strategien:

- **Greedy Decoding:** Wähle immer das wahrscheinlichste Token → langweilig, aber präzise
- **Temperature Sampling:** Höhere Temperatur = mehr Kreativität/Zufall
- **Top-k Sampling:** Wähle zufällig aus den Top k wahrscheinlichsten Tokens
- **Nucleus (Top-p) Sampling:** Wähle aus Tokens, die zusammen p% Wahrscheinlichkeit haben

Größenordnungen (GPT-3.5 / GPT-4):

- **Parameter:** 175 Milliarden (GPT-3) bis über 1 Billion (GPT-4)
- **Layers:** 96+ Transformer-Layer
- **Embedding-Dimension:** 12.288 (statt 768)
- **Kontext-Länge:** 8.000 - 128.000 Tokens
- **Inference-Zeit:** ~50-100ms pro Token

Zurück zur Anwendung

Zurück zu deiner Frage vom Anfang: „Was ist die Hauptstadt von Frankreich?“ Das Modell hat sie nicht nachgeschlagen und auch nicht verstanden — es hat sie in Tokens zerlegt, in Vektoren übersetzt, durch die Transformer-Schichten geschickt und am Ende für jedes mögliche nächste Token eine Wahrscheinlichkeit ausgerechnet. „Paris“ hat gewonnen, weil es in Milliarden Trainingssätzen nach genau dieser Wortfolge am häufigsten kam. Und dann fing dasselbe Spiel für das nächste Token wieder von vorn an, bis der Satz stand.

Wenn dir also ein Chatbot antwortet, siehst du ab jetzt keine Magie mehr, sondern eine Vorhersage-Maschine bei der Arbeit — und du weißt auch, warum sie sich manchmal mit voller Überzeugung irrt: Eine hohe Wahrscheinlichkeit ist keine Wahrheit. Das Herzstück dieser Maschine sind dabei ein paar wenige Rechenoperationen auf Tensoren, immer und immer wieder. Welche das genau sind, nimmst du dir als Nächstes in „Tensor-Operationen“ vor — danach kannst du eine Transformer-Schicht nicht nur beschreiben, sondern nachrechnen.

Zum Weiterlesen

- **Das Paper, das alles ausgelöst hat:** Ashish Vaswani u. a., *„Attention Is All You Need“*, 2017 — arxiv.org/abs/1706.03762 — hier wird die Transformer-Architektur aus Schritt 4 zum ersten Mal beschrieben.
- **Bebildert nachvollzogen:** Jay Alammar, *„The Illustrated GPT-2“* — jalammar.github.io/illustrated-gpt2 — er zeichnet genau die Pipeline, die du oben durchgeklickt hast.
- **Ein echtes Modell im Original:** Alec Radford u. a., *„Language Models are Unsupervised Multitask Learners“* (GPT-2), 2019 — PDF bei OpenAI (cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf) — mit den Größenordnungen, die im Abschnitt „Technische Details“ auftauchen.
- **Zum Mitbauen (Video, ca. 2 Stunden):** Andrej Karpathy, *„Let’s build GPT: from scratch, in code, spelled out“* — youtube.com (youtube.com/watch?v=kCc8FmEb1nY) — weil man das Prinzip am schnellsten begreift, wenn man es einmal selbst tippt.
- **Auf HerrKI:** 320 — Attention (wie das Modell entscheidet, worauf es schaut) und 350 — Halluzinationen (was passiert, wenn die wahrscheinlichste Antwort falsch ist).

Kurzer Check – mit Lösung

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

Drei schnelle Fragen, dann bist du durch — klick einfach deine Antwort an:

Frage 1: Was macht ein LLM im Kern, wenn es dir antwortet?

- Es sucht die fertige Antwort in einer Datenbank
- ✔ **Es sagt Token für Token das wahrscheinlich nächste Token vorher — bis die Antwort steht**
- Es kopiert den passendsten Satz aus dem Internet

Genau! Kein Nachschlagen, sondern fortlaufende Vorhersage: Frage-Tokens rein, dann wächst die Antwort Stück für Stück — jedes neue Token wird Teil der Eingabe für das nächste.

Frage 2: Was passiert im Transformer-Schritt?

- Der Text wird in eine andere Sprache übersetzt
- ✔ **Die Token-Vektoren werden miteinander verrechnet, damit jedes Token seinen Kontext kennt — die Shape bleibt dabei [Tokens, 768]**
- Die Tokens werden alphabetisch sortiert

Richtig — nur Matrix-Multiplikationen, keine Magie. Wie das Modell dabei entscheidet, worauf es schaut, zeigt dir das Attention-Kapitel. (Was eine Matrix-Multiplikation genau ist, siehst du gleich in „Tensor-Operationen“.)

Frage 3: Warum kann dieselbe Frage zweimal leicht verschiedene Antworten ergeben?

- Das Modell würfelt völlig zufällig
- ✔ **Die Ausgabe ist eine Wahrscheinlichkeits-Verteilung — beim Ziehen des nächsten Tokens fällt die Wahl mal anders aus**
- Der Server war beim zweiten Mal schneller

Genau: Am Ende steht für jedes mögliche Token eine Wahrscheinlichkeit. Zieht man nicht stur das Top-Token (Stichwort Temperature), variiert die Antwort — mehr dazu im Halluzinations-Kapitel.

Das nimmst du mit

- **Ein LLM sagt vorher, es schlägt nicht nach.** Es gibt keine Datenbank mit Antworten — nur die Frage „welches Token kommt als Nächstes?“.
- **Sechs Schritte, immer dieselben:** Tokenization → Token-IDs → Embeddings → Transformer → Output-Layer → Token für Token generieren.
- **Der Transformer verrechnet Kontext.** Jedes Token schaut auf alle anderen — die Shape `[Tokens, 768]` bleibt dabei gleich, nur die Zahlen ändern sich.
- **Am Ende steht eine Wahrscheinlichkeits-Verteilung** über das gesamte Vokabular, nicht ein fertiges Wort. Welches gezogen wird, steuert die Temperature.
- **Deshalb variieren Antworten — und deshalb gibt es Halluzinationen.** Wahrscheinlich und wahr sind zwei verschiedene Dinge.

Jetzt hast du den kompletten Kreis geschlossen: Frage → Tokens → Embeddings → Transformer-Mathe → Wahrscheinlichkeiten → Antwort, Token für Token. Wenn dir ChatGPT das nächste Mal antwortet, siehst du keine Magie mehr, sondern eine Vorhersage-Maschine bei der Arbeit - und du weißt, dass „Paris" nur deshalb kommt, weil es die 92-%-Hand hebt. Von hier aus hast du drei spannende Wege: baue selbst dein erstes Netz, schau der KI beim „Denken" zu (Reasoning) oder erkunde die Bedeutungs-Landkarte (Latent Space), auf der das alles spielt.

So zitierst du dieses Kapitel

HerrKI: Wie LLMs Fragen beantworten. herrki.de/050_wie_llms_antworten, ein privates Hobbyprojekt von Terragon.de, abgerufen am 09.09.2026.