KI FÜR ENTWICKLER

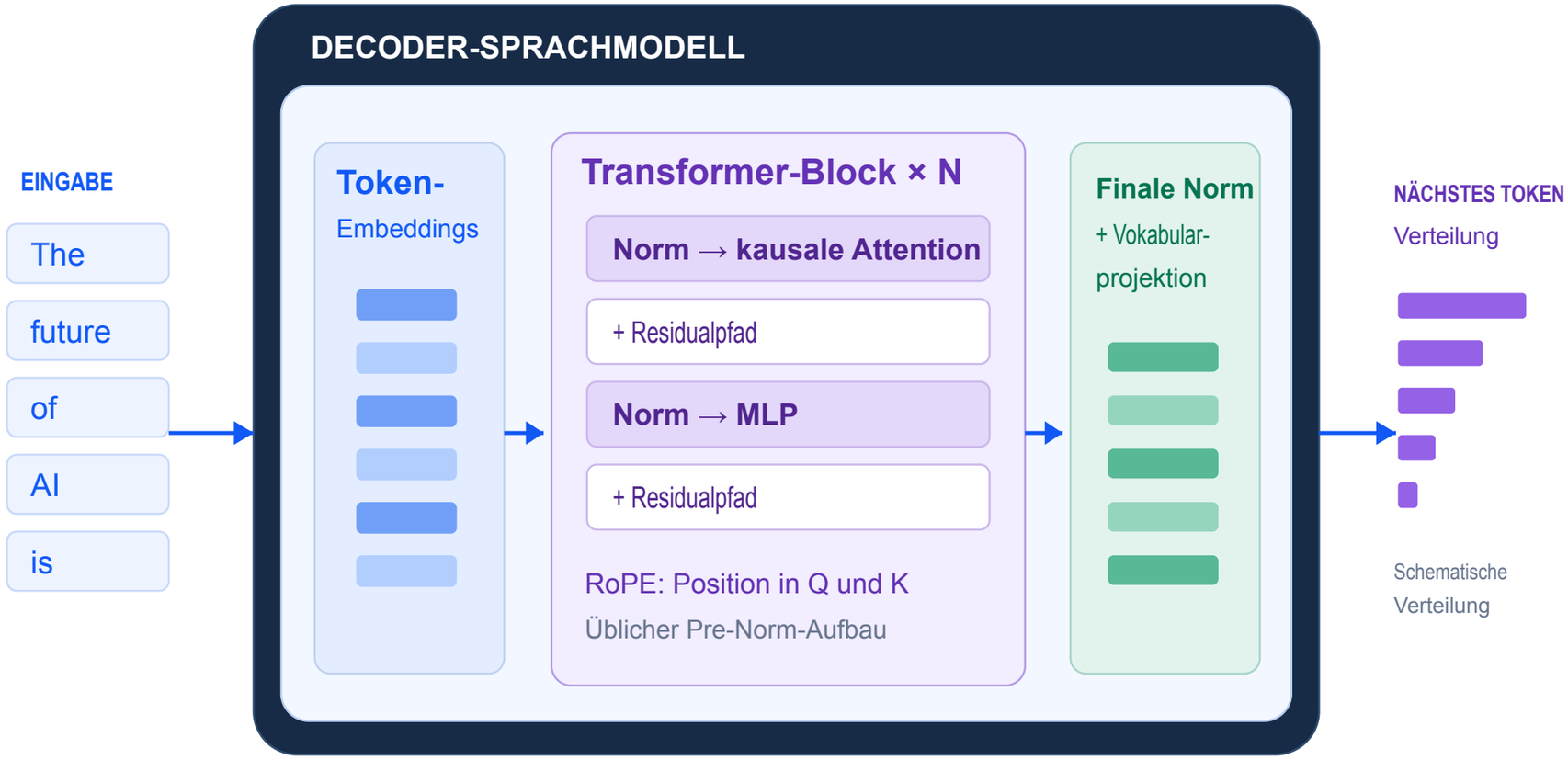
LLM-Grundlagen II
Im Modell

In Teil I war das Modell eine Blackbox.
Jetzt öffnen wir sie.

Hineinschauen
Tokens durch den Decoder-Stapel verfolgen.

Attention verstehen
Wie Kontext die nächste Token-Vorhersage prägt.

Intuition entwickeln
Architektur mit dem Entwickleralltag verbinden.

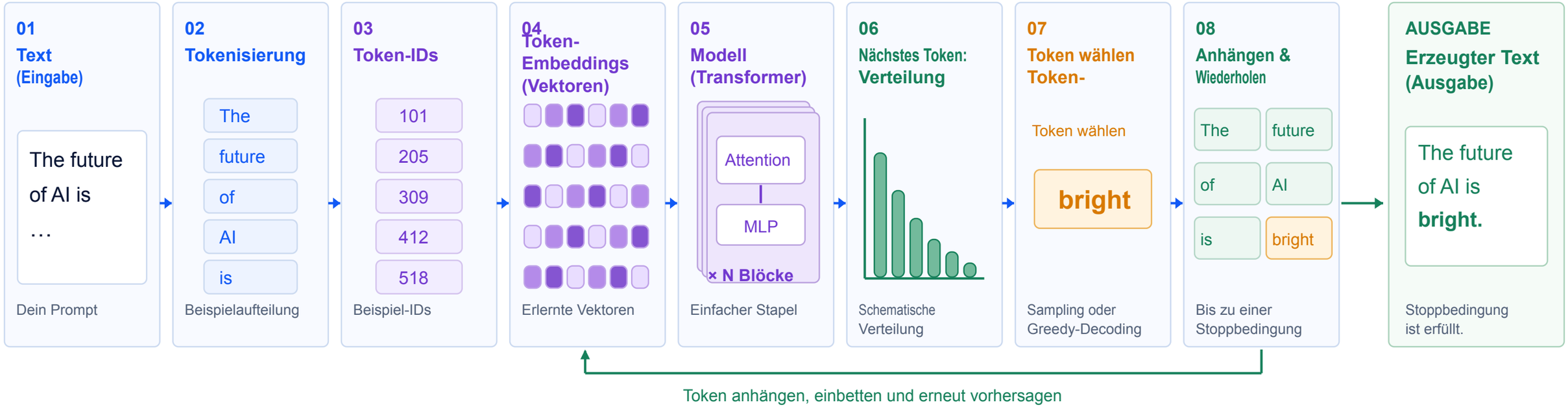


UNSER WEG Tokens → Darstellungen → Attention → Kontext → nächstes Token

Hugging Face · Llama-Dokumentation

Rückblick: Vom Text zur Vorhersage

Teil I zeigte den gesamten Weg vom Eingabetext zum nächsten Token.
Hier noch einmal das Gesamtbild.



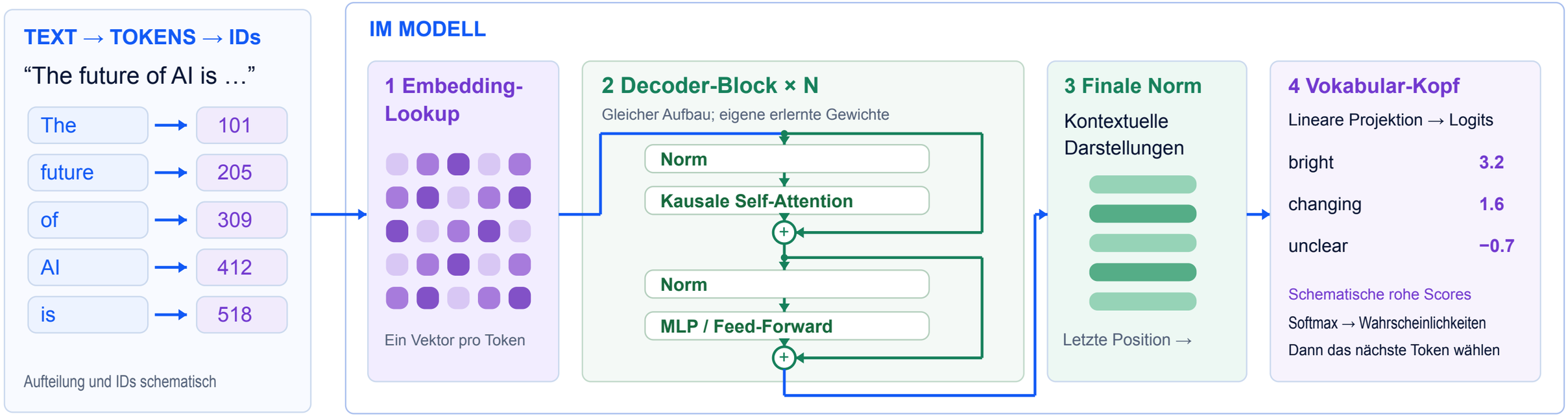
Kernaussage: LLMs verarbeiten keinen Text direkt, sondern Zahlenvektoren. Sie sagen das nächste Token vorher – Schritt für Schritt.

Jay Alammar · Der illustrierte GPT-2 Hugging Face · Textgenerierung

Was steckt in der Blackbox?

Ein Decoder-Transformer bildet aus Token-IDs kontextuelle Vektoren und nächste Token-Scores (Logits).

Hier: ein üblicher sequenzieller Pre-Norm-Aufbau, wie bei Llama-Modellen.



Rotary Position Embedding (RoPE) dreht Query- und Key-Vektoren gemäß der Tokenposition.
So reagiert Attention auf relative Positionen. Die kausale Maske sperrt künftige Tokens.

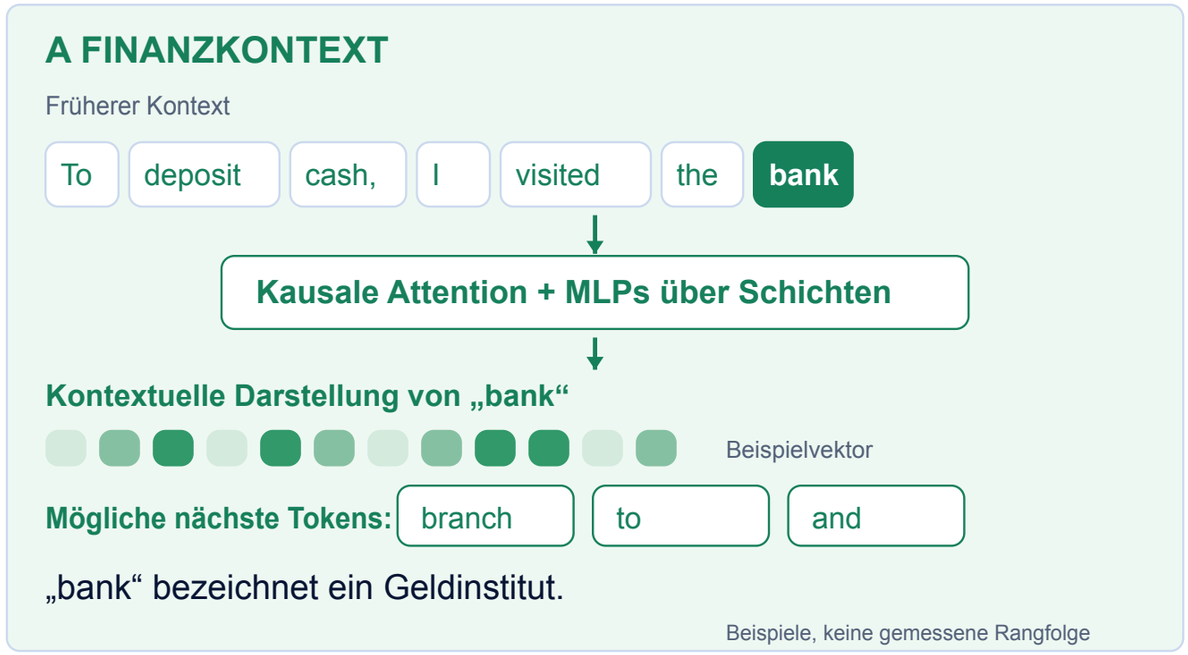
GRUNDIDEE Attention mischt Kontext, MLPs formen Merkmale und der Vokabular-Kopf bewertet nächste Tokens.

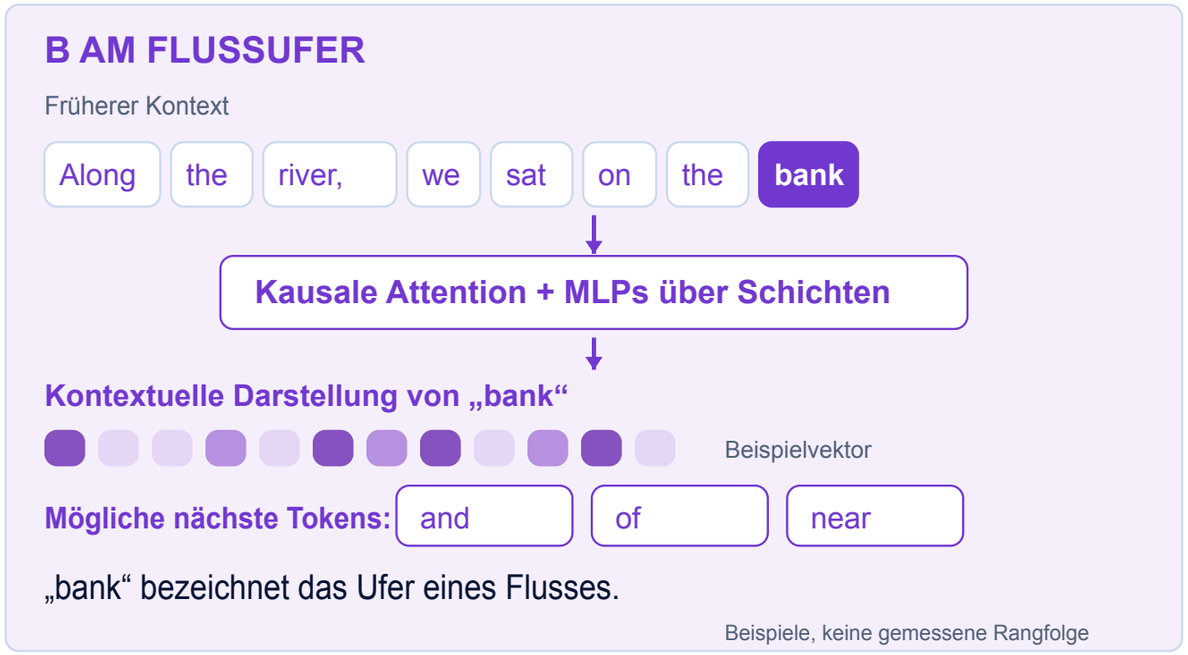
Touvron et al. · LLaMA (2023) Hugging Face · Llama-Architektur

Warum Kontext zählt

Dasselbe Token kann je nach Kontext andere Darstellungen und nächste Token-Vorhersagen haben.

Im kausalen Decoder nutzt jede Position sich selbst und frühere Tokens – niemals spätere.





GRUNDIDEE Die Token-ID bleibt gleich; seine Darstellung ändert sich mit dem vorhergehenden Kontext.

Jay Alammar · Der illustrierte GPT-2 Hugging Face · Kausale Sprachmodellierung

Der Transformer – Die Kernarchitektur

Viele moderne LLMs nutzen Decoder-Transformer: Wiederholte Blöcke verfeinern kontextuelle Darstellungen.

Gezeigt ist ein üblicher Pre-Norm-Aufbau. Block, Positionsmethode und Attention variieren je nach Modell.

WARUM DIESE ARCHITEKTUR?

Kontext verknüpfen

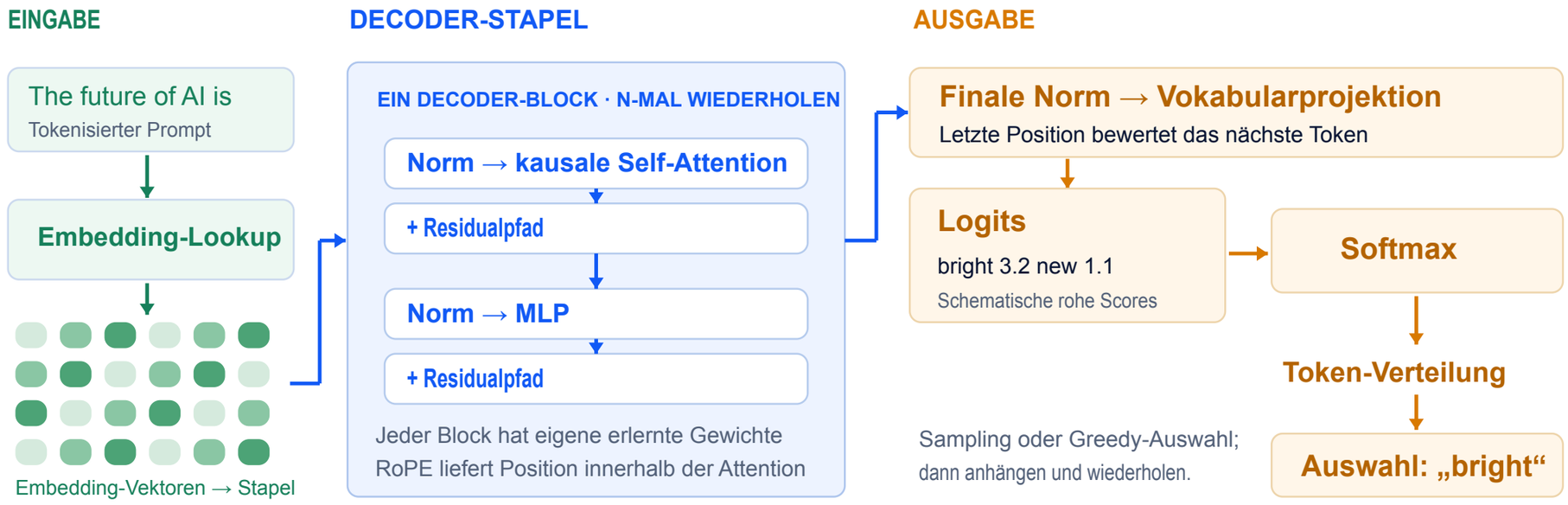
Kausale Attention verbindet Informationen erlaubter Tokens.

Tiefe + Residualpfade

Schichten verfeinern Merkmale; Skip-Verbindungen helfen im Training.

Parallel rechnen

Training / Prefill: parallel.
Decoding: schrittweise.



Attention mischt Positionen, MLPs formen Merkmale. Norm und Residualpfade stabilisieren den Stapel.

GRUNDIDEE Decoder-Blöcke bilden Kontext; der Ausgabekopf macht daraus Scores für das nächste Token.

Touvron et al. · LLaMA (2023) Hugging Face · Llama-Architektur

Der Transformer-Block

Ein Decoder-Block hat zwei Teilschichten: Attention mischt Kontext, das MLP formt Tokenmerkmale.

Sequenzieller Pre-Norm-Aufbau: vor jeder Teilschicht normalisieren, dann ihr Update zum Residualstrom addieren.



AUFGABEN DER BAUSTEINE

Attention

Jede Position nutzt sich und frühere Positionen für kontextuelle Merkmale.

Norm + Residualpfad

Eingabe normalisieren; Ausgabe zum unveränderten Residualpfad addieren.

MLP / Feed-Forward

Formt Merkmale unabhängig an jeder Position um.

Norm = Normalisierung. MLP = mehrschichtiges Perzeptron. Normierung und Aktivierung variieren je nach Modell.

PRE-NORM $h = x + \text{Attention}(\text{Norm}(x))$ $y = h + \text{MLP}(\text{Norm}(h))$

Touvron et al. · LLaMA (2023) Hugging Face · Llama-Architektur

Attention – Die Grundidee

Attention bildet je Tokenposition eine gewichtete Mischung der Informationen zugänglicher Positionen.

Gewichte hängen von Eingabe, Schicht und Attention-Kopf ab. Sie werden für jede Query neu berechnet.

BEISPIEL · „bank“ VERARBEITEN

To	deposit	cash,	I	visited	the	bank	yesterday

Frühere + aktuelle Tokens: erlaubt

Zukunft: maskiert

Attention-Gewichte für die Query bei „bank“



0.04	0.34	0.28	0.03	0.10	0.06	0.15	0.00

Hohes Gewicht → größerer Beitrag des Value-Vektors dieser Position.

Beispielgewichte eines Kopfes, keine gemessenen Modellausgaben. Die Summe ist 1.

SO ENTSTEHT DIE MISCHUNG

1 Vergleichen

Die aktuelle Query wird mit den Keys zugänglicher Positionen verglichen.

2 Normieren

Zukunft maskieren, dann Scores mit Softmax in Gewichte umwandeln.

3 Mischen

Jeden Value-Vektor gewichten und zu einem Ausgabevektor aufsummieren.

Queries, Keys und Values sind erlernte Projektionen der Token-Darstellungen. Dazu mehr auf den nächsten Folien.

GRUNDIDEE Attention mischt Value-Vektoren. Sie wählt kein nächstes Token und belegt keine Erklärung einer Antwort.

Vaswani et al. · Attention Is All You Need Jay Alammar · Der illustrierte GPT-2

Self-Attention in der Praxis

Folie 7 zeigte eine Query bei „bank“. Jetzt geschieht dasselbe an jeder Tokenposition.

Jede Zeile hat eine eigene Query, erlaubte Keys und Gewichte. Alle Positionen teilen die erlernten Projektionen.

WER KANN WEN BEACHTEN?

Query ↓ / Key →	To	deposit	cash,	I	visited	the	bank	yesterday
To		×	×	×	×	×	×	×
deposit			×	×	×	×	×	×
cash,				×	×	×	×	×
I					×	×	×	×
visited						×	×	×
the							×	×
bank	.04	.34	.28	.03	.10	.06	.15	0
yesterday								

Blau: erlaubt · Grau: maskiert · Violett: Beispielgewichte aus Folie 7

1 Eine Query je Position

„bank“ nutzt sich und frühere Tokens.
„yesterday“ kann auch „bank“ nutzen.
„bank“ sieht nicht voraus auf „yesterday“.

2 Eine kontextuelle Ausgabe je Position

Jede Zeile mischt zugängliche Value-Vektoren.
Die violette Zeile ergibt 1; die anderen
haben eigene, hier nicht gezeigte Gewichte.

3 Parallel rechnen, kausal zugreifen

Training und Prompt-Prefill berechnen
Zeilen parallel. Die Maske sperrt die Zukunft.

„Self“ heißt: Queries, Keys und Values stammen aus derselben Folge. Gezeigt: ein Kopf in einer Schicht.

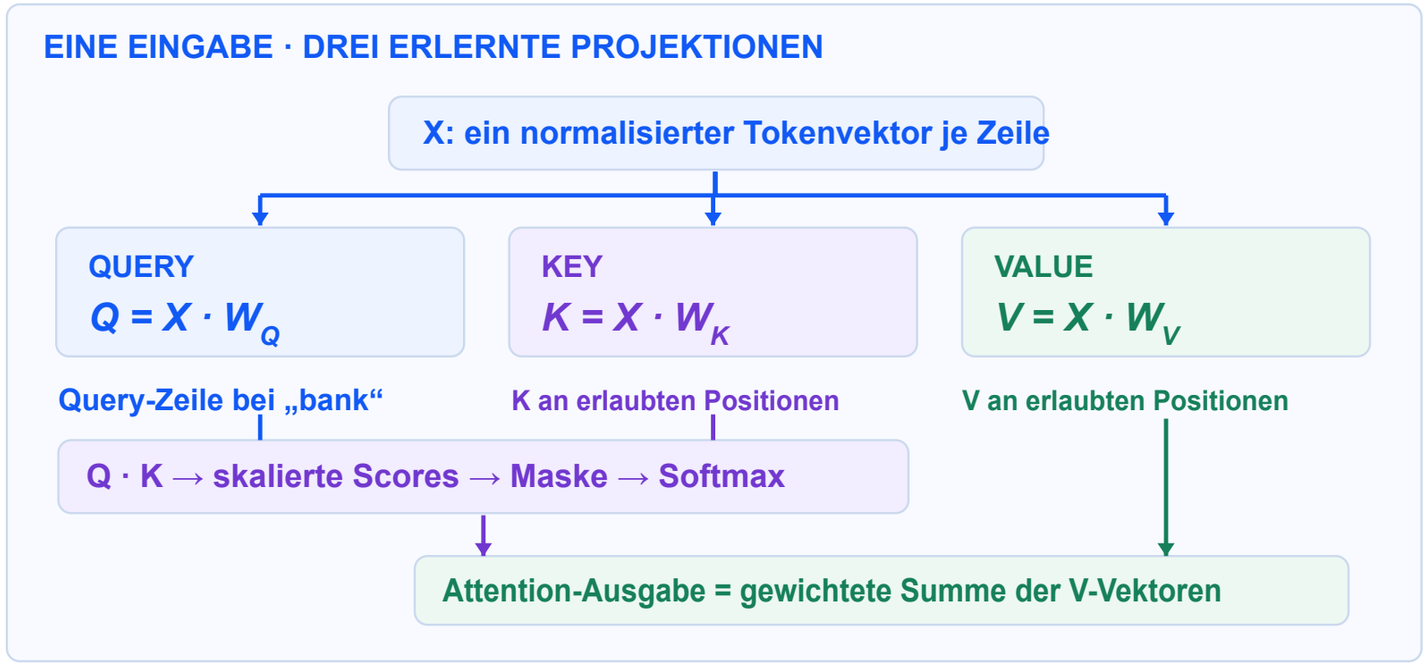
GRUNDIDEE Jede Position bildet ihre kontextuelle Darstellung – ohne künftige Tokens zu sehen.

Vaswani et al. · Attention Is All You Need Jay Alammar · Der illustrierte GPT-2

Queries, Keys und Values

Drei erlernte lineare Projektionen erzeugen Queries, Keys und Values aus denselben Eingabedarstellungen.

Weiter mit „bank“: Die Query wird mit erlaubten Keys verglichen; die Gewichte mischen dann die Values.



Query: Was passt zu dieser Position?

Q bestimmt bei „bank“ die Bewertung der verfügbaren Keys in diesem Kopf.

Key: Wie passt eine Position?

Jede Position liefert einen K-Vektor. RoPE dreht Q und K vor der Bewertung.

Value: Welche Information wird gemischt?

Jede Position liefert auch einen V-Vektor. Die Gewichte skalieren die Vektoren, dann werden sie zu einer Ausgabe addiert.

Ein Kopf dargestellt. X hat eine Zeile je Token; erlernte W-Projektionen werden über Positionen geteilt.

GRUNDIDEE Queries und Keys bestimmen Gewichte. Values liefern die zu mischenden Informationen.

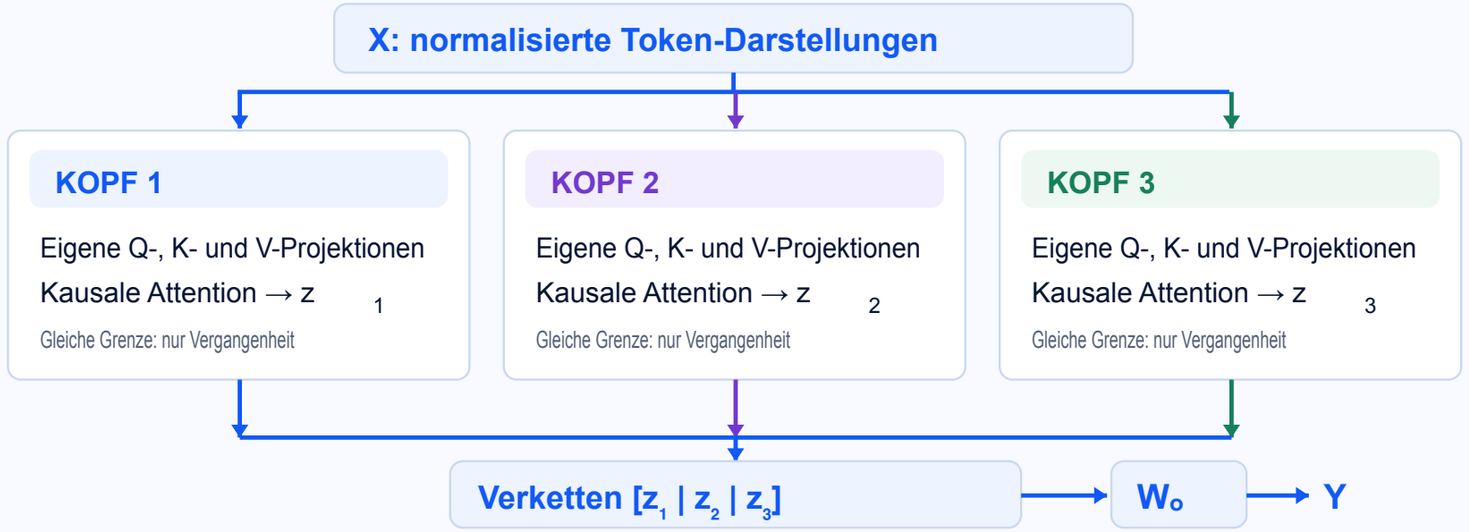
Vaswani et al. · Attention Is All You Need Jay Alammar · Der illustrierte GPT-2

Multi-Head-Attention

Mehrere Attention-Köpfe verarbeiten dieselbe Folge parallel mit verschiedenen erlernten Projektionen.

Ihre Ausgaben werden verkettet, projiziert und zur Residualaddition im Decoder-Block weitergereicht.

EINE FOLGE · MEHRERE PARALLELE KÖPFE



WAS DIE GRAFIK ZEIGT

1 Jeder Kopf hat erlernte Q/K/V-Projektionen.
Er kann andere Attention-Gewichte zuweisen.

2 Ausgaben werden verkettet und mit
der erlernten Ausgabeprojektion W_o gemischt.

3 Köpfe haben keine festen semantischen Rollen.
Muster und nützliche Rollen entstehen im Training.

MODELLVARIANTEN

MHA: eigene K/V je Query-Kopf.
GQA / MQA: K/V für Gruppen / alle Köpfe geteilt.

Zur Übersicht nur drei Köpfe. Kopfzahl und K/V-Teilung hängen von der Modellkonfiguration ab.

GRUNDIDEE Mehrere erlernte Sichten auf denselben Kontext bilden eine gemeinsame Attention-Ausgabe.

Vaswani et al. · Attention Is All You Need Ainslie et al. · Grouped-Query Attention Hugging Face · Llama-Dokumentation

Position zählt

Dieselben Wörter in anderer Reihenfolge können ein anderes Ereignis beschreiben.

Viele Decoder-Modelle nutzen RoPE, damit Attention die Position berücksichtigt.

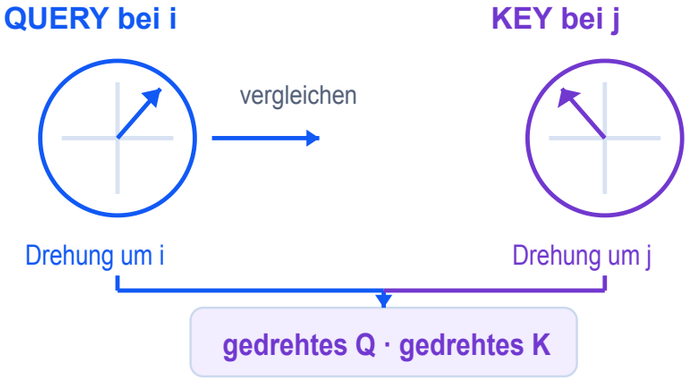
GLEICHE WÖRTER · ANDERE ROLLEN

POS	0	1	2	3	4	5
	The	dog	chased	the	cat	.
	Jäger: Hund				Gejagt: Katze	
	The	cat	chased	the	dog	.
	Jäger: Katze				Gejagt: Hund	

Wortboxen sind schematisch; ein Tokenizer kann Wörter in kleinere Teile zerlegen.

RoPE · ROTARY POSITION EMBEDDING

Für einen Attention-Kopf: Q und K nach Position drehen.



Der relative Abstand beeinflusst den Score.

Die kausale Maske sperrt separat künftige Positionen.

RoPE ist eine Positionsmethode (z. B. Llama). Andere Modelle nutzen gelernte Positionen oder Attention-Bias.

GRUNDIDEE Position hilft Attention zu erkennen, was ein Token enthält und wo es steht.

Su et al. · RoFormer / RoPE Hugging Face · Llama-Dokumentation

Schicht für Schicht

Teil I lieferte Token-Embeddings: erlernte Startvektoren für Token-IDs.

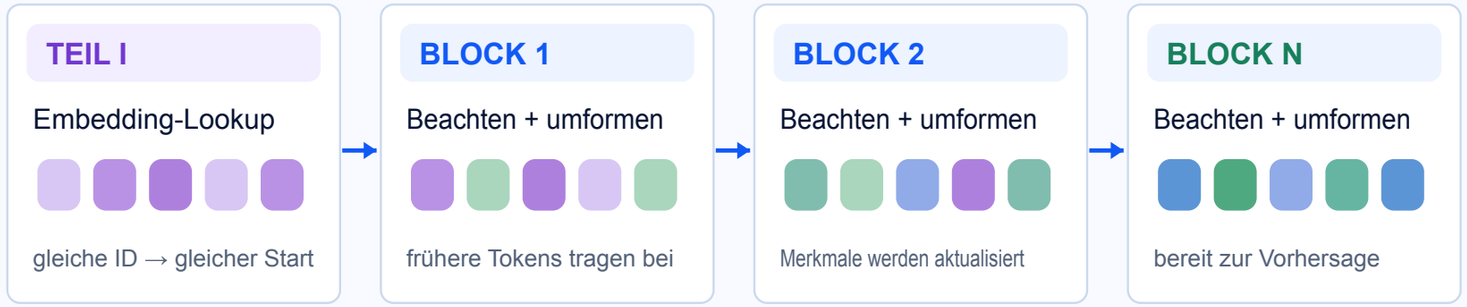
Decoder-Blöcke aktualisieren die Folge wiederholt und machen jede Position kontextabhängig.

VOM TOKEN-EMBEDDING ZUR KONTEXTUELLEN DARSTELLUNG

Beispielprompt · Wörter zur Übersicht

I	deposited	cash	at	the	bank	.

bank = Fokus



Nur „bank“ ist hervorgehoben. Jede Position durchläuft jeden Block; jeder Block hat eigene Gewichte.

WAS JEDER BLOCK TUT

1 Kausale Self-Attention

Mischt frühere und aktuelle Positionen.

2 MLP / Feed-Forward

Formt Merkmale an jeder Position um.

3 Residualpfade + Pre-Norm

Führen den Strom an beiden Teilschichten vorbei.

Danach: finale Norm → Vokabular-Scores (Logits).

Schichtrollen sind nicht fest: Es gibt keine allgemeine „Grammatik-“ oder „Faktenschicht“.

GRUNDIDEE Embeddings bilden den Start; gestapelte Blöcke erzeugen kontextsensitive Zustände zur Vorhersage.

Vaswani et al. · Attention Is All You Need Hugging Face · Llama-Architektur

Kontextfenster

Das Modell nutzt die Tokens dieser Anfrage, einschließlich relevanter früherer Nachrichten, falls mitgesendet.

Die Anwendung wählt die Inhalte; Chatrollen und Inhalte werden als modellspezifische Tokenfolge kodiert.

WAS KANN IM AKTUELLEN KONTEXT STEHEN?

System-/Entwickleranweisungen	falls genutzt
Frühere Nutzer- und Assistentenbeiträge	falls behalten
Tool-Ergebnisse / abgerufene Dokumente	falls geliefert
Aktuelle Nutzernachricht	neue Eingabe
Bisher erzeugte Assistenten-Tokens	bei der Ausgabe

ENDLICHES TOKEN-BUDGET

Bei vollem Fenster können ältere Inhalte entfallen oder zusammengefasst werden.

Die genaue Grenze hängt von Modell und Bereitstellung ab.

KONTEXTFENSTER ≠ LANGZEITGEDÄCHTNIS

Das Fenster ist die Arbeitseingabe dieser Anfrage.

Prompting allein verändert keine Modellgewichte.

Gespeicherte Fakten wirken nur, wenn die App sie mitgibt.

FÜR ENTWICKLER

1. **Ausgabe einplanen**
   Eingabe und erzeugte Tokens teilen die Grenze.
2. **Nützlichen Kontext wählen**
   Relevantes abrufen; alte Beiträge zusammenfassen.
3. **Verbleibende Inhalte prüfen**
   Kürzung testen und kritische Fakten verifizieren.

Ein größeres Fenster fasst mehr Tokens, garantiert aber keine zuverlässige Nutzung jedes Details.

GRUNDIDEE Deine App muss nötige Informationen auswählen, einpassen und aktuell halten.

Hugging Face · Chat-Vorlagen OpenAI · Kontextfenster Liu et al. · Lost in the Middle

Training und Inferenz

Dasselbe Decoder-Modell lernt aus bekannten Folgen oder erzeugt Text aus einem Prompt.

Training vergleicht mit bekannten nächsten Tokens; Inferenz ergänzt einen Prompt um neue Tokens.

TRAINING · GEWICHTE LERNEN

Bekannte Folge: „The future of AI is bright“

EINGABEPOSITIONEN

The	future	of	AI	is
↓	↓	↓	↓	↓
future	of	AI	is	bright

BEKANNTE ZIEL-TOKENS

Kausaler Vorwärtslauf → Scores an allen fünf Positionen

↓

Mit Zielen vergleichen → Verlust → Backprop → Gewichte ändern

Jede Position sieht nur ihren linken Kontext; viele Verluste laufen parallel.

INFERENZ · FESTE GEWICHTE NUTZEN

Prompt: „The future of AI is“

The	future	of	AI	is

↓

Prefill: Promptpositionen verarbeiten

↓

Nächste Token-Scores → ein Token wählen

↓

„bright“ anhängen → nächster Decoding-Schritt

Ausgabe-Tokens entstehen sequenziell; Gewichte bleiben fest.

Ein KV-Cache kann frühere Keys und Values wiederverwenden.

Wortboxen vereinfachen die Tokenisierung; „bright“ ist ein Beispiel, keine garantierte Vorhersage.

GRUNDIDEE Training lernt Gewichte aus Fehlern; Inferenz erzeugt mit festen Gewichten jeweils ein neues Token.

Hugging Face · Kausale Sprachmodellierung Hugging Face · KV-Cache

Warum LLMs falschliegen können

Eine flüssige Antwort kann unbelegte oder falsche Aussagen enthalten.

Token-Vorhersage erzeugt Text; faktische Richtigkeit braucht eigene Belege und Prüfungen.

WO FEHLER ENTSTEHEN

Frage + Kontext → Modell bewertet Tokens → Flüssige Antwort

FLÜSSIG ≠ GEPRÜFT Eine Aussage braucht mehr Belege als wahrscheinlich klingende Wörter.

1	**Erlernte Muster**	Trainingstexte können Fehler und Irrtümer enthalten.
2	**Fehlende Belege**	Nötige Fakten können fehlen, veraltet sein oder schlecht genutzt werden.
3	**Schlussfolgern / Generieren**	Berechnung, Schluss oder mehrstufige Antwort können scheitern.

Aufgaben und Modelle scheitern unterschiedlich; teste deinen eigenen Anwendungsfall.

ÜBERPRÜFBARE ANTWORTEN BAUEN

1. **Mit Belegen fundieren**
   Relevante, vertrauenswürdige Quellen abrufen.
2. **Quellen prüfen**
   Wichtige Aussagen mit der tatsächlichen Quelle abgleichen.
3. **Tools nutzen**
   Berechnungen, IDs und strukturierte Fakten validieren.
4. **Evaluieren + enthalten**
   Schwierige Fälle testen; „Ich weiß es nicht“ erlauben.

Eine Token-Wahrscheinlichkeit ist nicht die Wahrscheinlichkeit, dass eine Aussage wahr ist.

GRUNDIDEE Generierte Aussagen als Vorschläge behandeln: mit vertrauenswürdigen Belegen und Tools prüfen.

Lin et al. · TruthfulQA Lewis et al. · Retrieval-Augmented Generation Min et al. · FActScore

Kernaussagen & Ausblick

Wir haben Tokens von Embeddings über Decoder-Blöcke bis zu nächsten Token-Scores verfolgt.

Nutze dieses Verständnis – und seine Grenzen – beim Entwurf von KI-Anwendungen.

DAS MODELL IN VIER IDEEN

01 TOKENS → VEKTOREN

Teil I: Token-IDs werden erlernten Embeddings zugeordnet.

Sie sind die Startvektoren des Prompts.

02 SCHICHTEN BILDEN KONTEXT

Kausale Attention und MLPs aktualisieren Vektoren.

Normen und Residualpfade stützen den Stapel.

03 VORHERSAGEN & WIEDERHOLEN

Finale Zustände liefern nächste Token-Logits.

Token wählen, anhängen und fortfahren.

04 GRENZEN KENNEN

Kontext ist endlich; flüssiger Text kann falsch sein.

Belege und Überprüfung bleiben unverzichtbar.

Positionsmethoden, Kopfteilung und Blockdetails variieren. Prüfe die Modellkonfiguration.

NÄCHSTE SCHRITTE FÜR ENTWICKLER

1. **Ein Modell prüfen**
   Schichten, Köpfe, Positionsmethode, Kontextgrenze.
2. **Einen Prompt verfolgen**
   Token-IDs → Logits → gewählte Ausgabe-Tokens.
3. **Auf Zuverlässigkeit setzen**
   Relevante Belege, Tools und Validierung ergänzen.

Dann an realen Aufgaben und Fehlerfällen evaluieren.

Decoder-Modelle teilen Grundideen, sind aber nicht identisch implementiert.

GRUNDIDEE Verstehe den Mechanismus und gestalte Kontext und Prüfungen passend dazu.

Vaswani et al. · Attention Is All You Need Hugging Face · Kausale Sprachmodellierung Hugging Face · Llama-Dokumentation