KI FÜR ENTWICKLER

LLM-Grundlagen I

Vom Text zur Vorhersage

Verstehen, wie große Sprachmodelle aus Text intelligente Antworten erzeugen.

Ziel dieses Moduls

Die Grundlagen von LLMs verstehen – von der Texteingabe bis zur Vorhersage des nächsten Tokens.

Für Entwickler

Das nötige Verständnis aufbauen, um LLMs wirksam in realen Anwendungen einzusetzen.

Weiter

LLM-Grundlagen II – Im Modell: Transformer, Attention, Kontext und Grenzen.



Ein LLM erzeugt Text, indem es wiederholt **das nächste Token vorhersagt.**

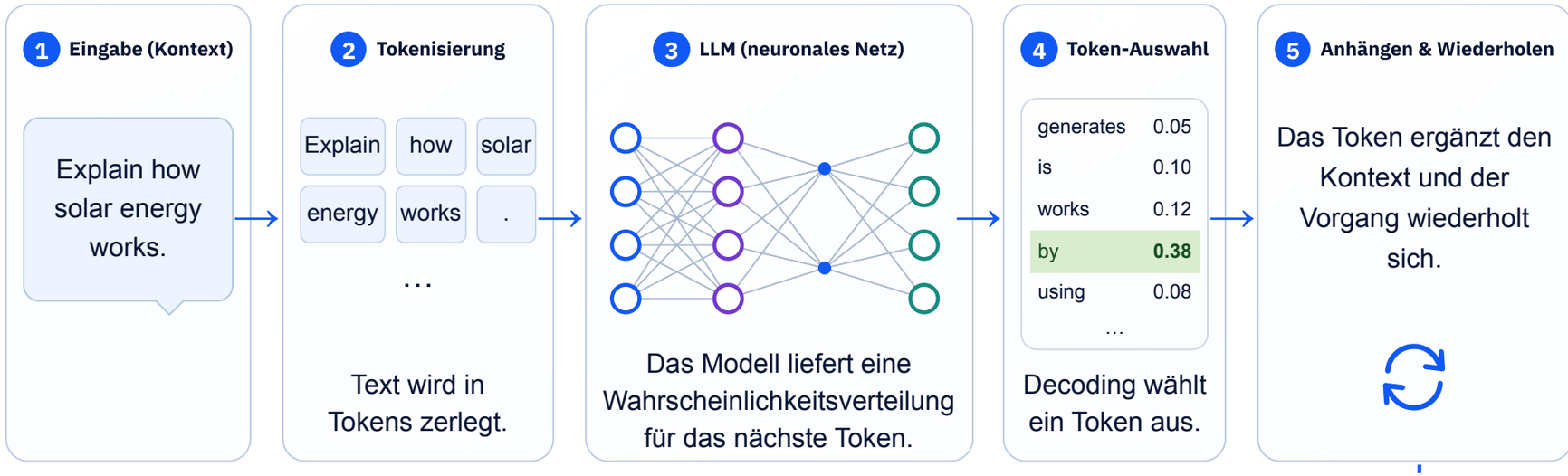
Was ist ein großes Sprachmodell?

Ein **LLM (Large Language Model)** ist ein neuronales Netz, das mit großen Textmengen trainiert wurde, um Sprache zu verarbeiten und zu erzeugen.

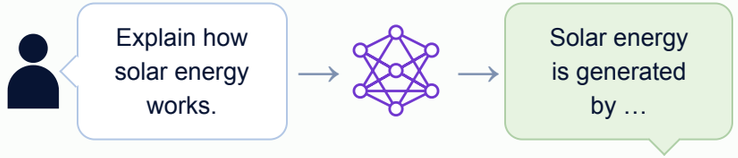
Für einen Kontext berechnet es Wahrscheinlichkeiten für das nächste Token. Decoding wählt ein Token und ergänzt den Kontext. Wiederholung erzeugt Antworten, Zusammenfassungen, Übersetzungen oder Code.

Ein LLM **ruft eine Antwort üblicherweise nicht als exakten Datenbankeintrag ab.** Es nutzt erlernte Muster; gespeicherte Trainingsinhalte können gelegentlich wiedergegeben werden.

SO ARBEITET EIN LLM (VEREINFACHT)



LLM: Erzeugt eine Antwort



Erzeugt Text aus erlernten Mustern und dem gegebenen Kontext. Kann gelegentlich gespeicherte Inhalte wiedergeben.

Datenbank: Ruft einen Eintrag ab



Liefert gespeicherte Informationen, die zu einer Abfrage passen.

In einem Satz:
Ein LLM erzeugt Sprache, indem es wiederholt eine Wahrscheinlichkeitsverteilung für das nächste Token vorhersagt und daraus ein Token auswählt.

Ein LLM allein ist keine vollständige KI-Anwendung

Ein LLM ist der Sprachmotor.
Eine KI-Anwendung ergänzt Daten, Tools, Abläufe, Sicherheit und eine gute Benutzeroberfläche.

LLM (Large Language Model)



Sprachmodell

Verarbeitet Kontext und sagt eine Verteilung für das nächste Token vorher.

+

Vollständige KI-Anwendung



Gesamtlösung

Verbindet das Modell mit weiteren Komponenten und schafft Nutzen.

Bausteine einer vollständigen KI-Anwendung



Daten & Wissen

Dokumente, Datenbanken, APIs und Fachwissen.



Retrieval & Suche

Relevante Informationen durch Suche, Indexierung oder RAG



Prompting & Orchestrierung

Anweisungen, Abläufe und Modellaufrufe koordinieren.



Tools & Integrationen

APIs, Geschäftssysteme, Rechner und Dienste



Leitplanken & Sicherheit

Datenschutz, Rechte, Validierung und Compliance steuern.



UI / UX

Nützliche Oberflächen für Web, Mobilgeräte und Chat gestalten.



Evaluation & Monitoring

Qualität, Kosten, Latenz, Nutzung und Feedback messen.



Kernaussage:
Ein LLM ist ein Baustein. Echter Nutzen entsteht durch die Verbindung mit Daten, Tools, Prozessen, Sicherheit und einer guten Benutzererfahrung.

Die Grundidee:
Das nächste Token vorhersagen

Im Kern jedes LLMs steht eine einfache Idee:
Aus Text vorhersagen, was als Nächstes folgt.

Die Antwort entsteht schrittweise.
Text entsteht Token für Token.

Beispiel

Prompt (Kontext):

The future of **renewable energy** is

Nächste Tokens (Wahrscheinlichkeiten)

bright	**0.68**
important	0.17
exciting	0.06
uncertain	0.03
…	…

Das Modell liefert eine Verteilung über das gesamte Vokabular.
Es wählt nicht die „richtige“ Antwort – es sagt vorher, was **am wahrscheinlichsten ist.**

So entsteht Text

1 Mit Kontext beginnen
The future of renewable energy is

2 Modell sagt nächstes Token vorher

Wahrscheinlichkeitsverteilung über alle Tokens

3 Nächstes Token wählen
bright
Auswahl anhand der Verteilung (z. B. Sampling)

4 Kontext ergänzen
The future of renewable energy is **bright**

5 Wiederholen
Wiederholen, bis eine Stoppbedingung gilt (z. B. Satzende)

Kernaussage:
LLMs erzeugen Text, indem sie wiederholt das nächste Token vorhersagen und den Kontext erweitern.

KI FÜR ENTWICKLER

Vom Text zu Tokens

LLMs zerlegen Text zur Verarbeitung in Tokens. Je nach Tokenizer stehen Tokens für **ganze Wörter, Wortteile, Satzzeichen, Leerraum** oder andere Bytefolgen.

LLMs verarbeiten Text anders als Menschen.
Sie verarbeiten Folgen von Tokens.

Beispiele für Tokenisierung

Ganze Wörter

Häufige Wörter können ein einzelnes Token bilden.

the | future | energy

Wortteile

Seltene oder komplexe Wörter können in Teilwörter zerfallen.

un | believ | able | micro | grid

Die genaue Aufteilung und Schreibweise hängen vom Tokenizer ab.

Satzzeichen

Satzzeichen können eigene Tokens bilden oder mit benachbartem Text zusammengefasst werden.

, | . | ! | ? | ; | : | (|) | “

Leerraum & Sonderzeichen

Leerzeichen und Zeilenumbrüche können eigene Tokens sein oder zu anderen Tokens gehören.

Leerzeichen | \n | \t

(schematisch)

Der Tokenisierungsvorgang (vereinfacht)

1 Eingabetext

Text eines Nutzers oder aus einem Dokument.

> The future of renewable energy is bright.

2 In Tokens zerlegen

Der Tokenizer zerlegt oder gruppiert Text nach seinem Vokabular und seinen Regeln.

The | future | of | renewable | energy | is | bright | .

3 Token-IDs

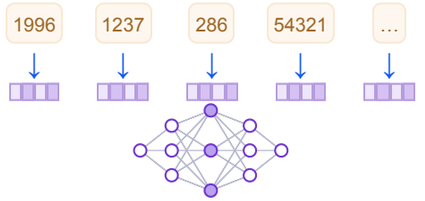
Jedes Token erhält eine ID im Vokabular dieses Tokenizers.

1996 | 1237 | 286 | 54321 | 12589 | 318 | 4567 | 13

Beispiel-IDs

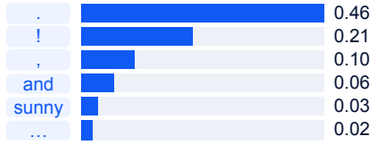
4 Embedding-Lookup

Token-IDs werden vor dem Transformer in erlernte Vektoren umgewandelt.



5 Modellverarbeitung

Der Transformer verarbeitet die Vektoren und sagt mögliche nächste Tokens vorher.



Nächste Token-Wahrscheinlichkeiten

Kernaussage:
Text → Tokens → Token-IDs → Embeddings → Transformer. Tokenisierung macht Text für ein LLM verarbeitbar und erzeugbar.

KI FÜR ENTWICKLER

Tokenisierung in der Praxis

Derselbe Satz kann je nach Modell und Tokenizer-Version unterschiedlich zerlegt werden.

Es gibt keine universell „richtige“ Tokenisierung.
Sie hängt vom **Vokabular und den Regeln des Tokenizers ab.**

Beispielsatz (Englisch)

> *The future of renewable energy is bright.*

Beispielhafte Token-Aufteilungen

Tokenizer A
Wortorientiertes Vokabular

The	future	of
renewable	energy	is
bright	.	

Beispiel: 8 Tokens

Tokenizer B
Anderes Vokabular

_The	_future	_of
_renew	able	_energy
_is	_bright	_.

Beispiel: 9 Tokens

Tokenizer C
Kleinere Teilwörter

_The	_fu	ture	_of
_rene	wa	ble	_en
ergy	_is	_bri	ght
_.			

Beispiel: 13 Tokens

Diese Beispiele zeigen nur das Prinzip. Exakte Tokens und IDs liefert nur die tatsächlich verwendete Tokenizer-Version.

Was passiert hier?

- Verschiedene Vokabulare erzeugen unterschiedliche Aufteilungen.
- Derselbe Satz kann unterschiedlich viele Tokens benötigen.
- Die Tokenzahl beeinflusst Kontextbedarf, Kosten und Latenz.
- Immer mit dem exakten Modell und der eingesetzten

Was beeinflusst die Tokenisierung?

Vokabular
Legt fest, welche Texteinheiten verfügbar sind.

Algorithmus
BPE, Unigram, WordPiece und verwandte Verfahren zerlegen Text unterschiedlich.

Sprache
Sprachen und Schriftsysteme können unterschiedlich tokenisiert werden.

Leerraum & Regeln
Leerzeichen, Satzzeichen, Großschreibung und Spezial-Tokens spielen eine Rolle.

Modell & Version
Die Tokenisierung kann sich zwischen Modellfamilien und Versionen ändern.

Selbst ausprobieren

Den Tokenizer des gewählten Modells verwenden.

```
tokenizer = load_tokenizer(model)
tokens = tokenizer.encode(text)
print(tokens)
print(len(tokens))
```

- ✓ Token-IDs
- ✓ Tokenzahl
- ✓ Dekodierte Tokens

Kernaussage: **Tokenisierung ist modellabhängig. Teste den eingesetzten Tokenizer, um Kontextgrenzen, Kosten und unerwartetes Verhalten im Griff zu behalten.**

KI FÜR ENTWICKLER

Warum Tokens zählen für Entwickler

Tokens sind mehr als ein technisches Detail.
Sie beeinflussen direkt **Verhalten, Leistung** und **Kosten** deiner KI-Anwendungen.

Jedes gesendete oder empfangene Token zählt.

Tokens verstehen. Besser entwickeln.

So beeinflussen Tokens deine Anwendung

1 Kontextgrenzen

Tokens bestimmen, wie viel das Modell gleichzeitig „sehen“ kann.



Belegt: **7.500 Tokens**
Grenze: **8.192 Tokens**
Beispielhaftes Kontextbudget

Warum wichtig:
Bei Überschreitung kann die Anfrage abgelehnt oder der Kontext gekürzt werden.

2 API-Kosten

LLM-APIs werden häufig nach Eingabe- und Ausgabe-Tokens abgerechnet.



Warum wichtig:
Mehr Tokens können mehr kosten. Eingabe und Ausgabe gezielt optimieren.

3 Latenz

Mehr Tokens bedeuten meist mehr Rechenarbeit und längere Antworten.



Warum wichtig:
Längere Prompts und Ausgaben können die Antwortzeit erhöhen.

4 RAG-Chunking

Daten für Retrieval in tokenbasierte Abschnitte teilen.



Warum wichtig:
Die Abschnittsgröße in Tokens beeinflusst Qualität und Relevanz der Suche.

Tokens sind eine echte Ressource

Eingabe-Tokens
Tokens im Prompt (System + Nutzer + Kontext).

Ausgabe-Tokens
Tokens, die das Modell als Antwort erzeugt.

Tokens gesamt
Eingabe + Ausgabe = Gesamtnutzung. Modellgrenzen beachten.

Rechenbeispiel

Prompt: 2.000 Tokens
Antwort: 1.000 Tokens
Gesamt: 3.000 Tokens

Kosten = 2 × Eingabepreis + 1 × Ausgabepreis
Preise je 1.000 Tokens

Empfehlungen für Entwickler

Kurz fassen
Unnötige Wörter, Wiederholungen und ausschweifenden Kontext entfernen.

Zusammenfassen
Lange Inhalte vor dem Senden zusammenfassen.

Früh filtern
Nur relevante Informationen in den Prompt aufnehmen.

Ausgabe steuern
Maximale Ausgabe-Tokens nach Möglichkeit begrenzen.

Messen & optimieren
Tokenverbrauch, Kosten und Latenz im Betrieb verfolgen.

Kernaussage:
Tokens sind eine begrenzte und wertvolle Ressource. Wer sie bei der Entwicklung berücksichtigt, baut schnellere, günstigere und and zuverlässigere KI-Anwendungen.

KI FÜR ENTWICKLER

Von Tokens zu Zahlen

LLMs arbeiten mit Zahlen statt mit Text.
Jedes Token erhält eine eindeutige Token-ID (eine Zahl).

Das Modell sieht keine Wörter, sondern Zahlenfolgen.
Zahlen hinein, Zahlen heraus.

Der Weg: Text → Tokens → Zahlen

1 Text (menschliche Sprache)

The future of renewable energy is bright.

Ein Mensch schreibt oder liefert Text

2 Tokens

The | future | of | renewable | energy | is | bright | . | (Ende)

Text wird in Tokens zerlegt (Wörter, Teile, Satzzeichen)

3 Token-IDs (Zahlen)

1996 | 1237 | 286 | 54321 | 12589 | 318 | 4567 | 13 | 2 (Ende)

Jedes Token erhält eine eindeutige Zahl (ID)

Tokens und IDs sind schematisch; Spezial-Tokens hängen vom Tokenizer ab.

So funktioniert es

1 Eingabetext

Du gibst Text ein (Prompt).

2 Tokenisierung

Text wird in Tokens zerlegt.

3 Token-ID-Zuordnung

Token	ID
The	1996
future	1237
of	286
…	…

Jedes Token erhält zu einer Zahl.

4 Modellverarbeitung

Das Modell verarbeitet numerische Darstellungen.

5 Ausgabe: IDs → Tokens → Text

54321 | 12589 | 318 | … | 2 (Ende)

↓

renewable energy is …

Ausgabe-IDs werden zurück in Tokens und Text umgewandelt.

Warum das wichtig ist

Effiziente Berechnung
Computer können Zahlen schnell verarbeiten.

Embeddings & Vektoren
Token-IDs dienen zum Nachschlagen erlernter Vektordarstellungen.

Basis der Generierung
Das Modell sagt die nächste Token-ID vorher, die wieder zu Text wird.

Eindeutige Zuordnung
Für ein gegebenes Vokabular erhält dasselbe Token stets dieselbe ID.

Kernaussage:
Tokens sind die Bausteine der Sprache, aber LLMs arbeiten mit Zahlen. Diese Brücke zu verstehen hilft beim Bau effizienter und zuverlässiger KI-Anwendungen.

KI FÜR ENTWICKLER

Von Token-IDs zu Vektoren

Token-IDs sind symbolische Indizes. Vor der Verarbeitung im Transformer wird jede ID einem erlernten Vektor zugeordnet: einem Token-Embedding.

Token-Embeddings bilden die **Ausgangsdarstellung.** Kontextabhängige Darstellungen entstehen im Transformer.

1 Token-IDs

Jede ID bezeichnet ein Token im Vokabular eines Tokenizers.

Schematisch

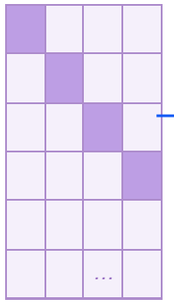
1996
1237
286
54321
…

Jede ID steht für ein Token.

2 Embedding-Lookup

Jede ID wählt eine Zeile der erlernten Embedding-Matrix.

1996 →
1237 →
286 →
54321 →
…



Embedding-Matrix
Vokabulargröße × Vektordimension

3 Token-Embeddings

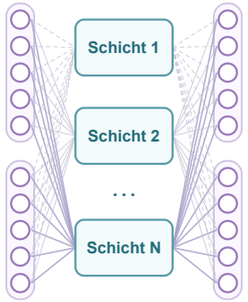
Jedes Token startet als dichter Vektor.

1996 → [0.21 −0.47 0.18 … 0.73]
1237 → [−0.11 0.62 −0.33 … 0.08]
286 → [0.74 0.05 −0.19 … −0.41]
54321 → [0.33 −0.22 0.91 … −0.12]
…

Im Training erlernt.

4 Transformer-Schichten

Attention und weitere Operationen verknüpfen Informationen über Tokenpositionen hinweg.



Informationen mischen sich über Tokens und Schichten.

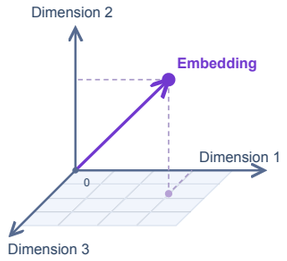
5 Kontextuelle Darstellungen

Der Vektor jeder Position hängt nun vom umgebenden Kontext ab.

1996 → [0.91 −0.13 0.67 … 0.22]
1237 → [−0.43 0.81 −0.26 … 0.12]
286 → [0.11 0.64 −0.72 … −0.36]
54321 → [0.57 −0.08 0.31 … −0.63]
…

Kontext beeinflusst die Darstellung jeder Position.

Was ist ein Embedding?



3D-Projektion · schematisch

Ein erlernter Punkt in einem hochdimensionalen Zahlenraum.

- Dimensionen werden erlernt, nicht von Hand vorgegeben.
- Ähnliche Verwendung kann zu ähnlichen Darstellungen führen.
- Anfängliche Token-Embeddings sind kontextunabhängig.

Gleiches Token, anderer Kontext

Kontext	Token-ID	Kontextuelle Ausgabe
“river bank”	8123	[0.82 −0.31 … 0.19]
“bank account”	8123	[−0.22 0.73 … 0.64]

Gleiche Token-ID und gleiches Start-Embedding.
Die kontextuelle Darstellung verändert sich.

IDs und Vektorwerte sind schematisch.

Wichtige Unterschiede

- **Token-ID** — symbolischer Index
- **Token-Embedding** — erlernter Startvektor
- **Kontextuelle Darstellung** — Ausgabe der Transformer-Schichten
- **Interne Berechnung** — numerische Vektoroperationen

Kernaussage:
Token-IDs werden erlernten Embeddings zugeordnet. Transformer-Schichten machen daraus kontextabhängige Darstellungen.

KI FÜR ENTWICKLER

Was geht in das Modell ein?

Nachrichten und bereitgestellter Kontext werden serialisiert, tokenisiert und vom Modell verarbeitet.

Generierungsparameter steuern die Auswahl der Ausgabe.

Das Modell kann nur Informationen aus seinen erlernten Parametern oder aus dem aktuellen Kontext nutzen.

Die Bestandteile des Modellkontexts

1 System-/Entwickler-Anweisungen

Legen Verhalten, Grenzen und Prioritäten fest.

Beispiel
Antworte klar und knapp.

+

2 Nutzer-Prompt

Die aktuelle Anfrage oder Frage.

Beispiel
Erkläre, wie Solarmodule funktionieren.

+

3 Kontext / Abgerufene Daten

Relevante Dokumente, Datensätze oder Tool-Ergebnisse für diese Anfrage.

Beispiel
Ausgewählter Inhalt aus einem technischen Dokument.

+

4 Gesprächsverlauf

Frühere Nachrichten für einen zusammenhängenden Dialog.

Beispiel
Frühere Diskussion über Installationskosten.

Serialisierte Nachrichten → Tokens → Transformer → Nächstes Token auswählen

Nur enthaltene Inhalte verbrauchen Kontextfenster-Tokens.

Generierungsparameter

Steuern Decoding – kein Teil des tokenisierten Prompts

- **Temperatur** — Verändert die Token-Verteilung.
- **Top-p / Top-k** — Begrenzt mögliche Tokens.
- **Max. Ausgabe-Tokens** — Begrenzt die Antwortlänge.
- **Stoppbedingungen** — Beenden die Generierung.

Der Ablauf

Nachrichten & Kontext → Serialisierung → Tokenisierung → Transformer → Token Wahrscheinlichkeiten → Decoding → Antwort

Warum das wichtig ist

- **Relevanz** — Aufgabenrelevante Informationen aufnehmen.
- **Priorität** — Nachrichtenrollen beeinflussen das Verhalten.
- **Zuverlässigkeit** — Relevanter Kontext kann unbelegte Antworten reduzieren, garantiert aber keine Richtigkeit.
- **Effizienz** — Unnötiger Kontext erhöht Tokenverbrauch, Latenz und Kosten.

Kernaussage:
Nachrichten und bereitgestellte Daten bilden den Modellkontext. Generierungsparameter steuern separat die Auswahl des nächsten Tokens.

KI FÜR ENTWICKLER

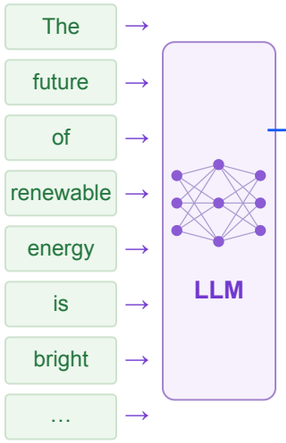
Vom Modell zu Token-Wahrscheinlichkeiten

Nach der Eingabeverarbeitung sagt das Modell voraus, welche Tokens als Nächstes wahrscheinlich sind. Es liefert eine **Wahrscheinlichkeitsverteilung** über das gesamte Vokabular.

Das Modell ordnet jedem möglichen Token eine Wahrscheinlichkeit zu. **Decoding wählt das nächste Token**

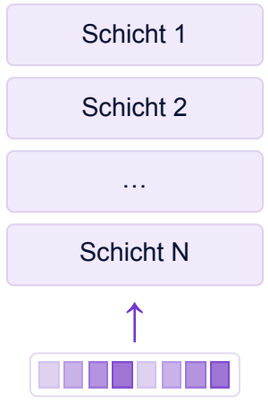
1 Eingabe ins Modell

Token-IDs werden zu Embeddings und dann vom Modell verarbeitet.



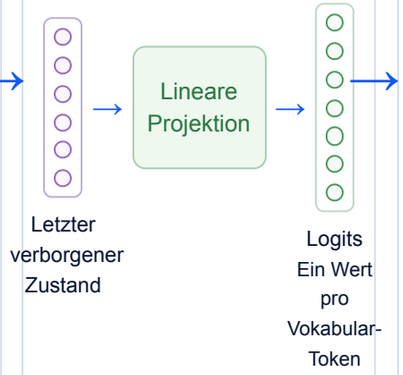
2 Modellverarbeitung

Das Modell verarbeitet die Folge mit Attention und Schichten.



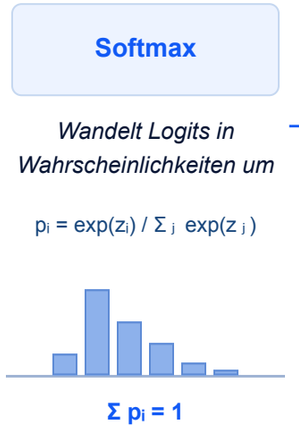
3 Nächste Token-Vorhersage

Der letzte verborgene Zustand wird auf Vokabular-Scores für das nächste Token projiziert.



4 Logits zu Wahrscheinlichkeiten

Softmax wandelt Logits in Wahrscheinlichkeiten um.



5 Wahrscheinlichkeitsverteilung

Jedes Token im Vokabular erhält eine Wahrscheinlichkeit. Nur wenige sind hoch.

Token	ID	Wahrscheinlichkeit
bright	5022	0.327
promising	8121	0.128
clean	3416	0.097
exciting	9910	0.056
the	1996	0.031
…	…	…
zzzzzzz	250111	0.000001

Beispielhafter Ausschnitt aus einem Vokabular mit 250.112 Tokens.

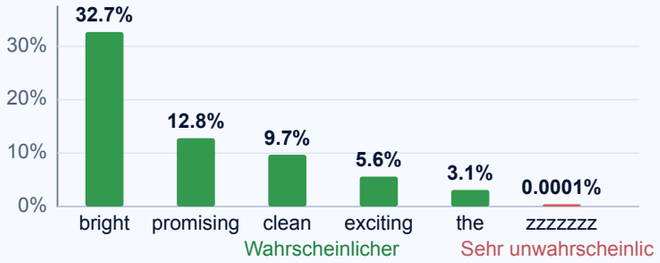
So sehen Wahrscheinlichkeiten aus

Beispielkontext:

The future of renewable energy is

Beispielwahrscheinlichkeiten

Verteilung über nächste Tokens



Top-k-Beispiel (k = 5)

Die fünf wahrscheinlichsten Tokens behalten:

1.	bright	0.327
2.	promising	0.128
3.	clean	0.097
4.	exciting	0.056
5.	the	0.031



Durch **0.639** teilen, neu normieren, dann ziehen.

Warum das wichtig ist

Das Modell sagt vorher, es weiß nicht.
Wahrscheinlichkeiten spiegeln erlernte Muster.

Generierung braucht eine Auswahl.
Decoding wählt oder zieht ein Token.

Entwickler steuern das Verhalten.
Temperatur, Top-k und Top-p formen die Auswahl.

Wahrscheinlichkeiten treiben Generierung.
Für Chat, Zusammenfassungen und Codevervollständigung.

Kernaussage:
Das Modell wandelt deine Eingabe in eine Verteilung über alle möglichen nächsten Tokens um. Diese Wahrscheinlichkeiten bilden die Grundlage jedes Generierungsschritts.

KI FÜR ENTWICKLER

Text erzeugen – Token für Token

LLMs erzeugen Text **autoregressiv**:
Sie sagen ein Token vorher, hängen es an die Folge an und wiederholen den Vorgang bis zu einer Stoppbedingung.

Das Modell baut die Antwort schrittweise auf, **Token für Token.**

So funktioniert ein einzelner Schritt

1 Aktuelle Eingabe (bisher)

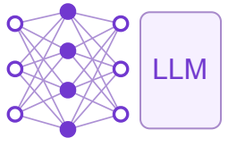
Die bisherige Folge ist die aktuelle Eingabe des Modells.

The future of renewable energy is bright

...

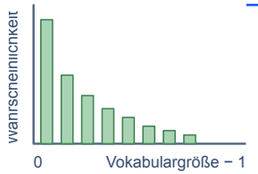
2 Modell verarbeitet die Eingabe

Das Modell bildet Embeddings und verarbeitet die Tokens.



3 Nächstes Token Token

Das Modell sagt eine Verteilung über das Vokabular vorher.



4 Token auswählen Token

Ein Token per Greedy-Decoding, Top-k oder einer anderen Strategie wählen.

.

Gewähltes Token (Satzende)

5 An die Folge anhängen

Das gewählte Token an die Folge anhängen.

The future of renewable energy is bright .

...

6 Wiederholen / Stoppen

Mit neuer Folge wiederholen, bis eine Stoppbedingung gilt.

Weiter erzeugen das nächste Token vorhersagt.

Die autoregressive Generierungsschleife

Start

Prompt

"The future of renewable energy is"

1. Verarbeiten — Aktuelle Eingabe verarbeiten.

2. Vorhersagen — Token-Verteilung vorhersagen.

3. Auswählen — Das nächste Token wählen.

4. Anhängen — Gewähltes Token anhängen.

Erzeugter Text

The future of renewable energy is **bright.**

Wiederholen bis zur Stoppbedingung:
- Ende-Token (EOS)
- Ausgabelimit erreicht · Stoppsequenz erkannt

Häufige Decoding-Strategien

Greedy (Top-1)
Stets das wahrscheinlichste Token wählen.



Top-k-Sampling
Aus den k wahrscheinlichsten Tokens ziehen.



Nucleus-Sampling (Top-p)
Aus der kleinsten Menge mit Gesamtwahrscheinlichkeit ≥ p ziehen.



Was steuert die Generierung?

Temperatur
Höher: vielfältiger.
Niedriger: fokussierter.

Top-k / Top-p
Begrenzt die berücksichtigten Kandidaten-Tokens.

Max. Ausgabe-Tokens
Begrenzt die Länge der erzeugten Antwort.

Stoppsequenzen
Tokens oder Zeichenfolgen, die die Generierung

Kernaussage:
Generierung ist eine Schleife: verarbeiten → vorhersagen → auswählen → anhängen → wiederholen. Jedes gewählte Token wird Teil der Eingabe für den nächsten Schritt.

KI FÜR ENTWICKLER

Wie wird das nächste Token ausgewählt?

Das Modell ordnet Tokens Wahrscheinlichkeiten zu. Eine **Decoding-Strategie** macht daraus ein einzelnes nächstes Token.

Unterschiedliche Strategien können **unterschiedliche Ergebnisse erzeugen.**

Von Wahrscheinlichkeiten zu einem Token

1 Modellausgabe

Das Modell liefert eine Verteilung über das Vokabular.

Token	Wahrscheinlichkeit
bright	0.327
promising	0.128
clean	0.097
exciting	0.056
the	0.031
…	…
zzzzzzz	0.000001

2 Decoding-Strategie anwenden

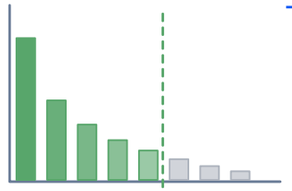
Die Strategie macht aus Wahrscheinlichkeiten eine Token-Auswahl.

Beispielstrategien

- Greedy (Top-1)
- Top-k-Sampling
- Nucleus (Top-p)
- Temperatur
- Wiederholungsstrafe

3 Filtern / Anpassen

Mit den gewählten Parametern Wahrscheinlichkeiten anpassen oder Kandidaten filtern.



Beispiel: **Top-k** behält nur die besten 5 Tokens.

4 Ziehen oder Wählen

Ein Token aus der Kandidatenverteilung wählen.



Deterministisch bei Greedy-Decoding, zufällig bei Sampling.

5 Nächstes Token

Das gewählte Token als nächstes Token der Folge verwenden.

Gewähltes Token

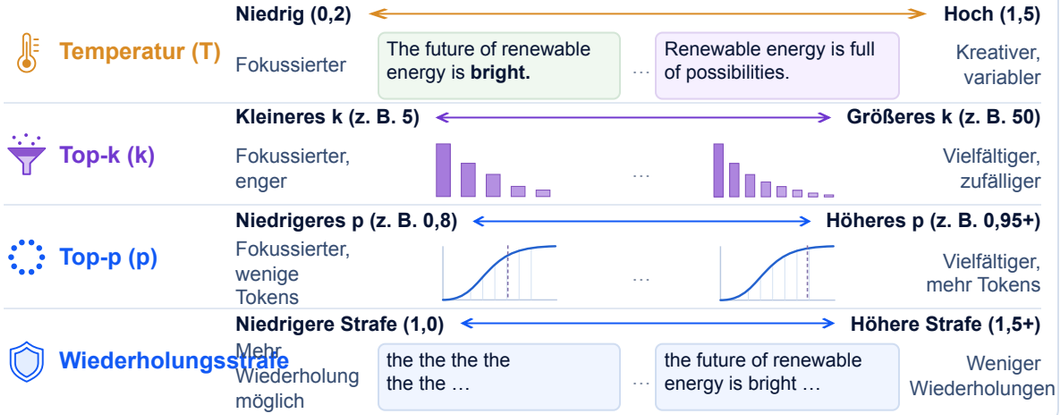
bright

Token anhängen und weiter generieren.

Decoding-Strategien im Überblick

Strategie	So funktioniert es	Parameter	Verhalten	Geeignet für
Greedy (Top-1)	Token mit höchster Wahrscheinlichkeit wählen.	—	Deterministisch; kann sich wiederholen.	Fokussierte und konsistente Ausgabe.
Top-k-Sampling	Aus den k wahrscheinlichsten Tokens ziehen.	k: 10, 20, 50	Kontrollierter Zufall; begrenzt seltene Tokens.	Ausgewogene Vielfalt und Kohärenz.
Nucleus (Top-p)	Aus der kleinsten Menge mit Gesamtwahrscheinlichkeit ≥ p ziehen.	p: 0.8, 0.9, 0.95	Adaptive Größe der Kandidatenmenge.	Offene Texte, Chat und Geschichten.
Temperatur	Logits vor Softmax skalieren.	T: 0.2–2.0	Niedriger: fokussierter. Höher: vielfältiger.	Vielfalt der Ausgabe steuern.
Wiederholungsstrafe	Scores bereits erzeugter Tokens anpassen.	z. B. 1,1–2,0	Wirkt Wiederholungen und Schleifen entgegen.	Längere Texte; Umsetzung variiert.

So beeinflussen Parameter die Ausgabe



Kernaussage:

Das Modell liefert Wahrscheinlichkeiten; Decoding macht daraus eine Entscheidung. Konsistenz, Vielfalt und Kreativität abstimmen – und faktische Richtigkeit gesondert prüfen.

KI FÜR ENTWICKLER

Was weiß das Modell wirklich?

Ein LLM ist **keine geprüfte Wissensdatenbank.**

Antworten entstehen aus erlernten Parametern, dem aktuellen Kontext und – sofern angebunden – externen Tools oder Retrieval.

Flüssige Antworten können trotzdem **unvollständig, veraltet oder falsch sein.**

Drei Quellen prägen eine Antwort

Erlernte Parameter

Im Training erlernte Muster.

- Sprach- und Codemuster
- Assoziationen und Lösungsstrategien
- Einige Fakten, unvollkommen abgebildet

Aktueller Kontext

Informationen in dieser Anfrage.

- Anweisungen und Gesprächsverlauf
- Dokumente oder abgerufene Passagen
- Beispiele und Tool-Ergebnisse

Externe Systeme

Optionale Quellen über Tools.

- Suche, Datenbanken und APIs
- Aktuelle oder firmenspezifische Daten
- Rechner und Geschäftssysteme

Erzeugte Antwort

Kombiniert erlernte Muster und gegebenen Kontext zur Token-Vorhersage.

Was nicht garantiert ist

- **Wahrheit** — Plausible Antworten können falsch sein.
- **Vollständigkeit** — Relevante Informationen können fehlen.
- **Aktualität** — Erlernte Parameter werden nicht laufend aktualisiert.
- **Quellentreue** — Das Modell kann Belege falsch lesen oder übergehen.
- **Konsistenz** — Ausgaben können zwischen Durchläufen variieren.

Retrieval-Augmented Generation (RAG)

Nutzerfrage → Retrieval → Gewählte Passagen → LLM-Kontext → Erzeugte Antwort

RAG liefert Belege für den Kontext. Dadurch wird eine Antwort nicht automatisch richtig oder aktuell.

Zuverlässige Anwendungen entwickeln

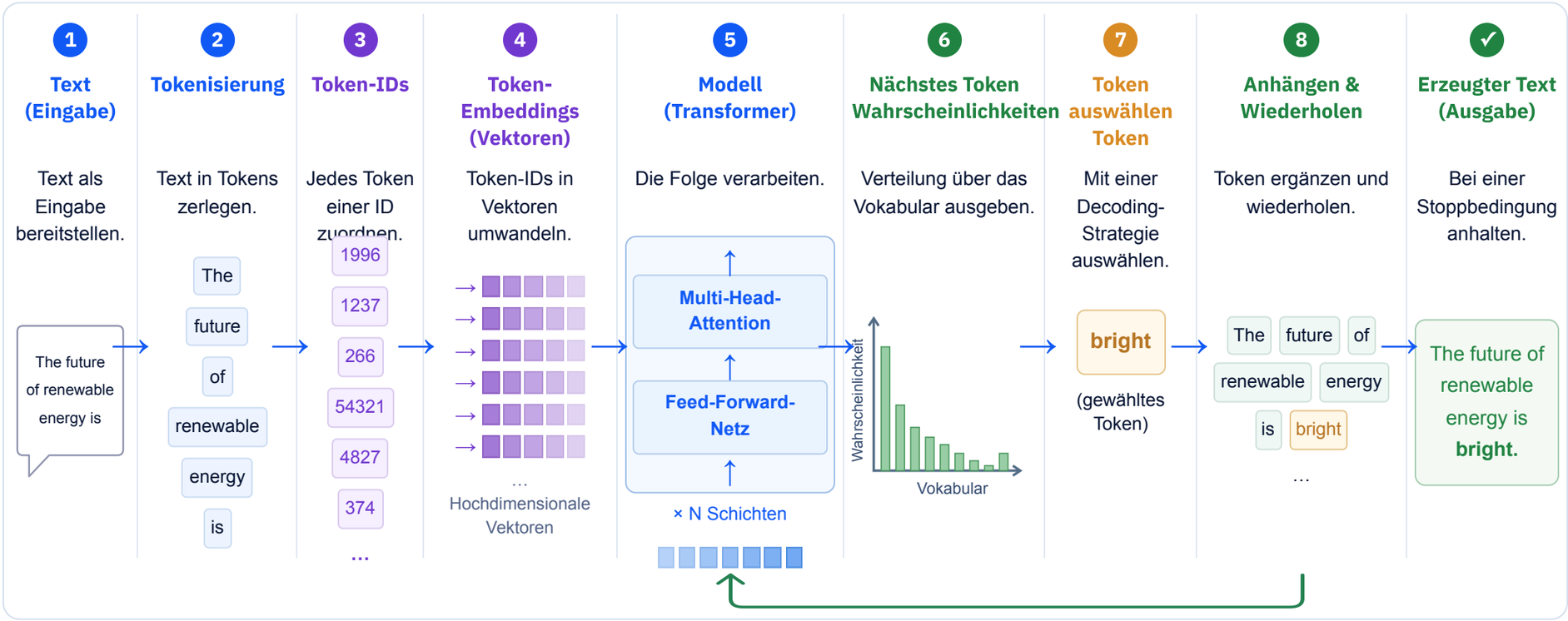
- Relevante und freigegebene Informationen abrufen.
- Tools für Berechnungen und aktuelle Daten nutzen.
- Bei Bedarf Quellen oder Belege anfordern.
- Wichtige Aussagen und Ausgaben validieren.
- Rechte, Monitoring und Evaluation einsetzen.

Kernaussage:
Ein LLM enthält erlernte statistische Darstellungen, keinen geprüften Faktenspeicher. Relevanten Kontext und Tools ergänzen – und wichtige Ergebnisse prüfen.

KI FÜR ENTWICKLER

Das Gesamtbild

Von deinem Text zur Modellantwort – jeder Schritt erklärt.



Die Grundidee

LLMs zerlegen Text in Tokens, verarbeiten sie als Vektoren, sagen das nächste Token vorher und erzeugen Text **autoregressiv.**

Was passiert im Modell?

- Attention gewichtet relevante Tokens.
- Schichten bilden kontextuelle Darstellungen.
- Die finale Ausgabe bewertet jedes mögliche nächste Token.

Zentrale Begriffe im Rückblick

- **Tokens:** Kleine Textbausteine.
- **Embeddings:** Vektoren, die Tokens darstellen.
- **Transformer:** Neuronales Netz zur Kontextverarbeitung.
- **Autoregressiv:** Ein Token nach dem anderen.

Warum das für Entwickler wichtig ist

- Modellverhalten verstehen.
- Parameter gezielter einsetzen (Temperatur, Top-k usw.).
- Antworten untersuchen und bewerten.
- Zuverlässige, effiziente KI-Anwendungen bauen.

Das Grundprinzip

LLMs sagen das nächste Token aus **erlernten Mustern und dem aktuellen Kontext vorher** — Schritt für Schritt.

Diese einfache Schleife ermöglicht leistungsfähige Systeme.

Kernaussage:
Der gesamte Ablauf ist eine Schleife: tokenisieren → einbetten → verarbeiten → vorhersagen → auswählen → anhängen → wiederholen. Token für Token entsteht die vollständige Antwort.

KI FÜR ENTWICKLER

Kernaussagen & Ausblick

Du weißt jetzt, wie LLMs Text in Tokens zerlegen, verarbeiten und Token für Token neuen Text erzeugen.

Merke: Vorhersagen hängen von erlernten Mustern **und dem aktuellen Kontext ab.**

Kernaussagen

1 **Text wird zu Tokens**

Die Eingabe wird in kleine Textbausteine zerlegt.

2 **Tokens erhalten IDs**

Jedes Token erhält eine ID im Vokabular.

3 **IDs werden zu Vektoren**

Token-IDs werden erlernten, verarbeitbaren Embeddings zugeordnet.

4 **Das Modell verarbeitet**

Ein Transformer verarbeitet Kontext mit Attention und Schichten.

5 **Wahrscheinlichkeiten vorhersagen**

Das Modell liefert eine Verteilung über das gesamte Vokabular.

6 **Das nächste Token auswählen**

Eine Decoding-Strategie wählt das nächste Ausgabe-Token.

7 **Anhängen und wiederholen**

Token ergänzen und bis zur Stoppbedingung wiederholen.

8 **Die fertige Ausgabe entsteht**

Die Folge gewählter Tokens bildet die erzeugte Antwort.

So geht es in dieser Reihe weiter

II **LLM-Grundlagen II**

Attention, Positionskodierung, Kontextfenster, Training und Inferenz sowie Modellgrenzen.

Fokus: Architektur & Interna

›

III **Embeddings & Semantische Suche**

Text-Embeddings, Vektorähnlichkeit, semantische Suche und RAG.

Fokus: Bedeutung & Retrieval

›

IV **Entwickeln mit LLMs**

Prompt Engineering, Tool-Nutzung, Function Calling und zuverlässige LLM-Anwendungen.

Fokus: Praktische Fähigkeiten

›

V **Produktion & bewährte Verfahren**

Evaluation, Sicherheit, Beobachtbarkeit, Kostenoptimierung und skalierter Betrieb.

Fokus: Zuverlässigkeit & Skalierung

Weiterentwickeln:
Je mehr du mit LLMs baust, desto besser verstehst du ihre Stärken und Grenzen. Experimentieren, verbessern und weiterlernen!