Schritt 6: Die Output-Schicht — Vom Vektor zum Wort

Tokenizer ✓ → Embedding ✓ → Pos. Enc. ✓ → MH-Attention ✓ → Add&Norm + FFN ✓ → **Output**

Erlebnispark — Der letzte Raum: *Alle Tokens haben 12 Blöcke durchlaufen. Sie sind jetzt voll mit Kontext und Wissen. Aber der Transformer muss eine* **Entscheidung** *treffen: Was ist das nächste Wort?*

Nur das **letzte Token** *wird befragt. In unserem Fall:* **„Matte"***. Sein Vektor enthält jetzt die gesamte Geschichte des Satzes — komprimiert in 4 Zahlen.*

Jetzt muss der Transformer aus diesen 4 Zahlen eine Entscheidung für eines von **50.000 möglichen Tokens** *machen.*

Warum nur das letzte Token?

Bei der Textgenerierung fragt GPT immer: *„Was kommt nach dem letzten Wort?"*

Der Satz „Die Katze sitzt auf der Matte" hat 6 Tokens. Das letzte Token „Matte" hat durch die Masked Attention **alle** vorherigen Tokens gesehen. Sein Vektor enthält den gesamten Satz-Kontext — er ist der informierteste.

Nur dieser letzte Vektor geht in die Output-Schicht.

Die 3 Schritte zum nächsten Wort

Letzter Vektor (4D) nach 12 Blöcken

↓

Projektion: 4D → 6 Logits (1 pro Wort im

Vokabular)

↓

Softmax: Logits → Wahrscheinlichkeiten (summieren zu 100%)

↓

Auswahl: Token mit höchster Wahrscheinlichkeit wählen

> **In Wirklichkeit:** GPT-2 hat ein Vokabular von ~50.257 Tokens. Die Projektion geht also von 768D auf 50.257 Zahlen — eine riesige Matrix!
>
> **Bei uns:** 4D → 6 Logits (unser Vokabular: Die, Katze, sitzt, auf, der, Matte).

Schritt 1: Projektion — Von 4D auf 6 Logits

> **Logit** = ein rohes Score-Wert für jedes mögliche nächste Token. Hoher Logit → das Wort ist wahrscheinlicher. Noch keine Prozente — das kommt mit Softmax.
>
> Die Projektion ist eine einfache Matrix-Multiplikation: `Logits = Vektor × W_out`

> **Trick:** In vielen Transformern ist W_{out} die **gleiche Matrix wie die Embedding-Tabelle — nur transponiert!** Das spart Parameter.
>
> Das ergibt Sinn: Die Embedding-Tabelle sagt „Token 1 (Katze) hat Vektor [0.0, 0.9, 0.1, 0.2]". Die Output-Projektion fragt umgekehrt: „Welches Token-Embedding passt am besten zu meinem Vektor?"
>
> Mathematisch: **Logit für Token i = Skalarprodukt(Vektor, Embedding von Token i)**

Unser letzter Vektor (nach 12 Blöcken) für „Matte". Da wir nur 1 Block gerechnet haben, nehmen wir einen realistischeren Beispiel-Vektor, der zeigt was nach vielen Blöcken passieren könnte:

Beispiel-Vektor nach 12 Blöcken: [-0.2, 0.1, 0.5, 0.8]

(Hypothetisch — der echte Wert käme aus 12× Attention+FFN)

Embedding-Tabelle (von Schritt 2) als Output-Projektion verwenden:

Token (ID)	Embedding [d_1, d_2, d_3, d_4]	Skalarprodukt mit [-0.2, 0.1, 0.5, 0.8]	Logit
Die (0)	[0.9, 0.1, 0.0, 0.1]	(-0.2×0.9) + (0.1×0.1) + (0.5×0.0) + (0.8×0.1)	
Katze (1)	[0.0, 0.9, 0.1, 0.2]	(-0.2×0.0) + (0.1×0.9) + (0.5×0.1) + (0.8×0.2)	
sitzt (2)	[0.0, 0.1, 0.9, 0.0]	(-0.2×0.0) + (0.1×0.1) + (0.5×0.9) + (0.8×0.0)	
auf (3)	[0.5, 0.0, 0.3, 0.4]	(-0.2×0.5) + (0.1×0.0) + (0.5×0.3) + (0.8×0.4)	
der (4)	[0.9, 0.1, 0.0, 0.1]	(-0.2×0.9) + (0.1×0.1) + (0.5×0.0) + (0.8×0.1)	
Matte (5)	[0.0, 0.0, 0.0, 0.9]	(-0.2×0.0) + (0.1×0.0) + (0.5×0.0) + (0.8×0.9)	

Rechnen wir aus:

```
Die:    -0.18 + 0.01 + 0.00 + 0.08 = -0.09
Katze: 0.00 + 0.09 + 0.05 + 0.16 = 0.30
sitzt: 0.00 + 0.01 + 0.45 + 0.00 = 0.46
auf:    -0.10 + 0.00 + 0.15 + 0.32 = 0.37
der:    -0.18 + 0.01 + 0.00 + 0.08 = -0.09
Matte: 0.00 + 0.00 + 0.00 + 0.72 = 0.72
```

Token	Die	Katze	sitzt	auf	der	Matte
Logit	-0.09	0.30	0.46	0.37	-0.09	**0.72**

Schritt 2: Softmax — Logits werden Wahrscheinlichkeiten

> Softmax kennst du schon von der Attention! Gleiche Formel:
> $P(\text{Token } i) = e^{\text{logit\_i}} / \Sigma\, e^{\text{logit\_j}}$

$e^{-0.09} = \mathbf{0.914}$ (Die)

$e^{0.30} = \mathbf{1.350}$ (Katze)

$e^{0.46} = \mathbf{1.584}$ (sitzt)

$e^{0.37} = \mathbf{1.448}$ (auf)

$e^{-0.09} = \mathbf{0.914}$ (der)

$e^{0.72} = \mathbf{2.054}$ (Matte)

Summe = 0.914 + 1.350 + 1.584 + 1.448 + 0.914 + 2.054 = **8.264**

Wahrscheinlichkeiten:

Die: 0.914 / 8.264 = **0.111** (11.1%)

Katze: 1.350 / 8.264 = **0.163** (16.3%)

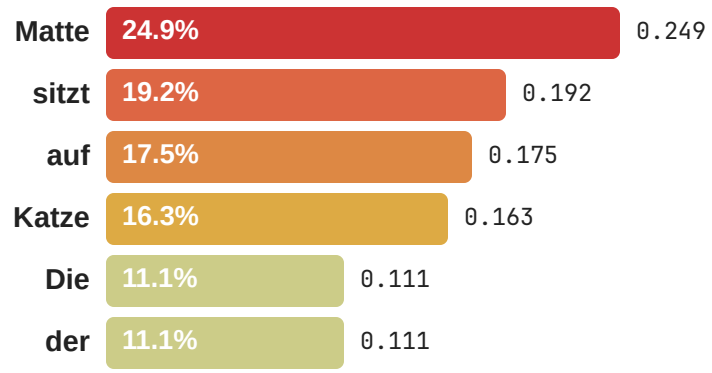
sitzt: 1.584 / 8.264 = **0.192** (19.2%)

auf: 1.448 / 8.264 = **0.175** (17.5%)

der: 0.914 / 8.264 = **0.111** (11.1%)

Matte: 2.054 / 8.264 = **0.249** (24.9%)

Visualisiert:



> **„Matte" hat die höchste Wahrscheinlichkeit (24.9%)!**

> Das klingt wenig? In Wirklichkeit wäre das mit einem echten Vokabular von 50.000 Tokens ein extrem starkes Signal. Stell dir vor: Ein Token hat 25%, während 49.999 andere sich die restlichen 75% teilen müssen — im Schnitt nur 0.0015% pro Token.

Schritt 3: Auswahl — Welches Token wird genommen?

Hier gibt es **verschiedene Strategien**:

Strategie	Wie?	Ergebnis bei uns
Greedy	Immer das wahrscheinlichste Token nehmen	Matte (24.9%)
Sampling	Zufällig wählen, gewichtet nach Wahrscheinlichkeit	Matte (25% Chance), sitzt (19% Chance), ...
Top-k	Nur aus den k wahrscheinlichsten wählen	Top-3: {Matte, sitzt, auf}
Top-p (nucleus)	Wähle aus den Top-Tokens bis Summe ≥ p	Top-p=0.6: {Matte, sitzt, auf} (24.9+19.2+17.5=61.6%)

Temperature — Kreativität einstellen

Erlebnispark: *Bevor Softmax die Logits in Wahrscheinlichkeiten verwandelt, kann man an einem* **Regler** *drehen: der* **Temperature**.

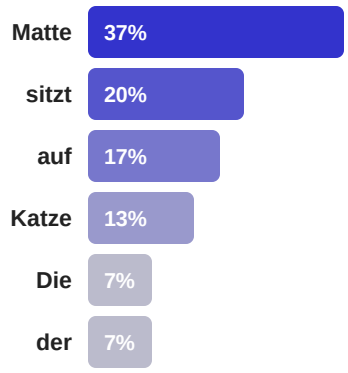
Niedrig = fokussiert, vorhersehbar. Hoch = kreativ, überraschend.

Formel: $P(\text{Token } i) = e^{(\text{logit}_i / T)} / \Sigma\, e^{(\text{logit}_j / T)}$

Die Logits werden einfach durch T geteilt, bevor Softmax angewendet wird.

T = 0.5 (Kalt = Fokussiert)

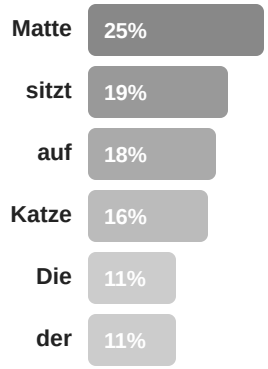
```
Logits / 0.5 → verdoppelt!
Matte: 0.72/0.5 = 1.44
sitzt: 0.46/0.5 = 0.92
Die: -0.09/0.5 = -0.18
```



Matte dominiert noch stärker

T = 1.0 (Normal)

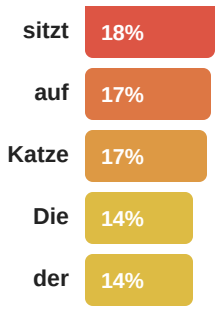
```
Logits / 1.0 → keine Änderung
(Das ist unser Ergebnis
von der vorherigen Seite)
```



Ausgeglichen

T = 2.0 (Heiß = Kreativ)

```
Logits / 2.0 → halbiert!
Matte: 0.72/2.0 = 0.36
sitzt: 0.46/2.0 = 0.23
Die: -0.09/2.0 = -0.045
```



Fast gleichverteilt — alles möglich

Zusammenfassung Temperature:

T → 0: Immer das wahrscheinlichste Token (= Greedy). Wiederholbar, langweilig.

T = 1: Originalverteilung. Ausgewogen.

T → ∞: Gleichverteilung. Komplett zufällig, sinnlos.

Die meisten Chat-Modelle nutzen T zwischen **0.7 und 1.0**.

Das große Bild: So generiert GPT Text

Erlebnispark — Die Endlosschleife: *Du hast gesehen wie EIN nächstes Wort vorhergesagt wird. Aber GPT schreibt ganze Sätze, Absätze, Aufsätze!*

Das Geheimnis: **Es hängt das vorhergesagte Wort an und macht alles nochmal.** *Und nochmal. Und nochmal. Token für Token.*

Autoregressive Generation — Schritt für Schritt:

1
`Input: "Die Katze sitzt auf der Matte"`
→ Ganzer Transformer (12 Blöcke) → Letzter Vektor → Softmax
→ Vorhersage: **"."** (Punkt)

2
`Input: "Die Katze sitzt auf der Matte ."`
→ Ganzer Transformer (12 Blöcke) → Letzter Vektor → Softmax
→ Vorhersage: **"Sie"**

3
`Input: "Die Katze sitzt auf der Matte . Sie"`
→ Ganzer Transformer (12 Blöcke) → Letzter Vektor → Softmax
→ Vorhersage: **"schnurrt"**

...
`Input: "Die Katze sitzt auf der Matte . Sie schnurrt ..."`
→ Immer weiter, bis ein Stop-Token kommt oder das Limit erreicht ist.

Beachte: Bei jedem Schritt wird der **gesamte** Transformer nochmal durchlaufen — nicht nur für das neue Token, sondern für den gesamten bisherigen Text. Jedes Token durchläuft alle 12 Blöcke von Anfang an.

Das ist teuer! Ein Satz mit 100 Tokens erfordert 100 komplette Durchläufe. Deshalb gibt es den **KV-Cache**: Die Key- und Value-Vektoren der bereits verarbeiteten Tokens werden gespeichert und wiederverwendet. Nur das neue Token muss komplett berechnet werden.

Übung: Temperature selbst berechnen

Berechne die Wahrscheinlichkeiten mit **T = 0.5**.
Teile dazu jeden Logit durch 0.5, bevor du Softmax anwendest.

✏️ Selbst rechnen

Logits: Die=-0.09, Katze=0.30, sitzt=0.46, auf=0.37, der=-0.09, Matte=0.72

Logits / 0.5:
Die: -0.09/0.5 = ___ Katze: 0.30/0.5 = ___ sitzt: 0.46/0.5 = ___
auf: 0.37/0.5 = ___ der: -0.09/0.5 = ___ Matte: 0.72/0.5 = ___

e^x: ___, ___, ___, ___, ___, ___ → Summe = ___

Wahrscheinlichkeiten:
Die: ___% Katze: ___% sitzt: ___% auf: ___% der: ___% Matte: ___%

Wie viel % hat Matte jetzt? ___ (Vergleiche mit T=1.0: 24.9%)

Zusammenfassung: Der komplette Transformer

Erlebnispark — Ende der Reise: *Du bist als Buchstabe auf ein Fließband gestiegen, wurdest zu einem Token geschnitten, hast eine Nummer bekommen, einen Vektor aus dem Regal gezogen, eine Position auf den Rücken gestempelt, 12 Mal einen Raum voller Brillen betreten, 12 Mal allein in einem stillen Raum nachgedacht, und am Ende hat man dich befragt: „Was kommt als Nächstes?"*

Das ist alles. Das ist ein Transformer.

#	Schicht	Papier-Blatt	Kernidee
1	**Tokenizer**	Blatt 1 + BPE	Text → Nummern
2	**Embedding**	Blatt 2	Nummern → Bedeutungs-Vektoren
3	**Pos. Encoding**	Blatt 3	Sinus/Cosinus addieren → Position codiert
4a	**Self-Attention**	Blatt 4a	Q·K → Softmax → gewichtete Summe V
4b	**Multi-Head + Mask**	Blatt 4b	Mehrere Perspektiven, nur rückwärts schauen
5	**Add & Norm + FFN**	Blatt 5	Skip Connection, Stabilisierung, allein nachdenken
6	**Output**	Blatt 6 (dieses)	Logits → Softmax → nächstes Wort wählen

Du hast einen Transformer aus Papier gebaut!

Was du jetzt weißt:

Konzept	Du kannst es erklären	Du kannst es berechnen
Tokenizer (BPE)	✓	✓
Embedding + Ähnlichkeit (Skalarprodukt)	✓	✓
Positional Encoding (Sinus/Cosinus)	✓	✓
Self-Attention (Q, K, V)	✓	✓

Softmax	✓	✓
Masked Attention (Dreiecks-Maske)	✓	✓
Multi-Head Attention	✓	✓
Skip Connection + Layer Norm	✓	✓
Feed-Forward Network + ReLU	✓	✓
Output-Projektion + Temperature	✓	✓
Autoregressive Generierung	✓	

Entwickelt im Gespräch. Nicht aus einem Lehrbuch. Aus dem Erlebnispark.

David & Claude, April 2026