Nachschlagen statt rechnen: N-Gramm-Speicher im Modell

Kapitel online

Ein Sprachmodell rechnet heute selbst für Fakten, die man einfach wüsste. DeepSeeks Engram gibt dem Modell ein Regal: Die letzten paar Wörter werden zur Zeilennummer, die Zeile ist ein gelernter Vektor, und weil die Adresse vorab bekannt ist, darf das Regal im Arbeitsspeicher stehen statt auf der Grafikkarte. Mit Hash-Tabelle zum Anfassen, U-Kurve der Aufteilung und Bandbreiten-Rechner.

Aktuell · Kapitel 32 von 50 · ca. 25 Min. · 300 XP

Einfach gesagt: Fragst du nach etwas Feststehendem wie „Guten Morgen“, muss die KI nicht rechnen — sie schlägt es in einem Regal voller gelernter Antworten nach.

Frage: **Was ist die Hauptstadt von Frankreich?** Du hast nicht gerechnet, nichts abgewogen — du *wusstest* es. Und wenn dich jemand mit „Guten …“ anspricht, leitest du das „Morgen“ nicht her: Es liegt in einem Fach in deinem Kopf, und du greifst hinein.

Ein Sprachmodell macht das anders. Für jedes Token läuft dieselbe Maschinerie: alle Schichten, alle Aufmerksamkeits-Köpfe (Kapitel 320), die volle Rechnung. Ob es „Paris“ ergänzt oder einen verschachtelten Beweis führt — **der Aufwand ist derselbe**.

Anfang 2026 hat DeepSeek dazu ein Paper veröffentlicht. Die Idee heißt **Engram** und ist erfrischend altmodisch: **Gib dem Modell ein Regal.** Die letzten paar Wörter werden zur Zeilennummer, dort liegt ein gelernter Vektor, und der wird dazuaddiert. Kein Rechnen. Nachschlagen.

HerrKI-Tipp: Ein Satz für dieses Kapitel — **„Wissen, das immer gleich ist, muss man nicht jedes Mal neu ausrechnen.“**

Was du hier lernst

- Warum es **zwei Sorten Wissen** gibt — und nur eine davon muss man rechnen
- Wie aus den **letzten Wörtern eine Zeilennummer** wird (Hashing, Kollisionen, Köpfe)
- Warum das **kein RAG** ist, obwohl beides „Wissen holen“ heißt
- Die **U-Kurve**: wie viel Regal und wie viel Rechnen zusammen am besten sind
- Warum die Tabelle im **billigen Arbeitsspeicher** liegen darf statt im teuren VRAM

Für den Unterricht

Einstieg (5 Min): Kapitel gemeinsam lesen, Alltagsbeispiel aufgreifen.

Erkunden am Gerät (15 Min): die interaktiven Module auf herrki.de/335_ngram_speicher ausprobieren.

Sichern mit Quiz (5 Min): „Kurzer Check" besprechen, „Das nimmst du mit" festhalten.

Zwei Sorten Wissen: Rechnen und Nachschlagen

„Conditional Memory via Scalable Lookup: A New Axis of Sparsity for Large Language Models“
Xin Cheng, Rui Tian, Wangding Zeng, Damai Dai u. a. (21 Autoren), **DeepSeek-AI**, eingereicht **12. Januar 2026**, überarbeitet Juli 2026 — arXiv:2601.07372, Code Apache-2.0. ⚠ **Forschungsarchitektur**: keine veröffentlichten Gewichte, nach unserem Wissensstand (September 2026) in keinem ausgelieferten Modell.

Ein Sprachmodell braucht zweierlei. **Rechenwissen**: erschließen, wer mit „sie“ gemeint ist — das braucht Tiefe und Abwägung. Und **Nachschlagewissen**: Auf „Guten“ folgt „Morgen“, nach „Sehr geehrte Damen und“ kommt „Herren“ — das hängt fast nur an den letzten zwei bis drei Wörtern. Heute wird beides gleich teuer behandelt.

Die erste Achse der Sparsamkeit: MoE

An einer Stelle sparen große Modelle längst — beim **Rechnen**: **Mixture of Experts** (MoE, „Mischung von Fachleuten“). Statt eines großen Rechenblocks je Schicht gibt es viele kleine **Experten-Teilnetze**, und ein winziges Netz davor — der **Router** — schickt jedes Token nur zu wenigen. Das Modell *hat* viele Parameter, aber je Token ist nur ein Bruchteil **aktiv**: **„conditional computation“**. Das Paper fragt: Warum ist das die *einzige* Achse, an der wir sparen?

Die zweite Achse: bedingtes Nachschlagen

Engrams Antwort: **„conditional memory“**, bedingtes Erinnern. Neben die Experten tritt eine große, **statische Tabelle** gelernter Vektoren, aus der je Token ein paar Zeilen geholt werden. Sie rechnet nichts, sie *ist* einfach da — die **„zweite Achse der Sparsamkeit“**.

Das Bild dazu: MoE ist die passende Fachperson im Raum — du *rechnest* weiter, nur gezielter. Engram ist die **Einmaleins-Tabelle an der Wand**: Für 7 × 8 *guckst du hin*.

Die Begründung ist nicht bloß „billiger“: Die **frühen Schichten** verbringen viel Kapazität mit **lokaler Musterrekonstruktion** („local pattern reconstruction“) — sie bauen zusammen, was ohnehin feststeht. Nimmt man ihnen das ab, bleibt die Attention für **globale Zusammenhänge** frei.

Runter zum Tensor: Aus den letzten Wörtern wird eine Zeilennummer

Fünf Handgriffe liegen zwischen den letzten Wörtern und dem fertigen Vektor — keiner ist eine Matrixmultiplikation:

1. **Tokenizer-Kompression:** Token-IDs werden auf **kanonische IDs** projiziert (NFKC, Kleinschreibung u. a.) — „Frankreich“ und „FRANKREICH“ werden dieselbe ID, das effektive Vokabular schrumpft um **23 %**.
2. **Suffix-N-Gramme:** die **letzten n kanonischen Tokens**, n bis 3 — also 2- und 3-Gramme; der Kontext ist absichtlich winzig.
3. **Multi-Head-Hashing:** je Ordnung **K Hash-Köpfe** (im Modell K = 8); jeder rechnet **deterministisch** einen Index in seine eigene Tabelle $E_{n,k}$ mit **Primzahlgröße** M. Zeilenlänge **1280**.
4. **Gating:** aus dem Hidden State h_t wird attention-ähnlich ein Skalar $\alpha_t \in (0,1)$; passt die Erinnerung nicht zum Kontext, geht α gegen 0. Danach eine kleine **Faltung** (Kernelgröße 4).
5. **Einbau:** addiert wird in **Schicht 2 und 15**. Die Faltungsparameter starten auf **null** — der Block ist anfangs die Identität. Einbettungen lernen mit Adam, **5-facher Lernrate**, ohne Weight Decay.

Am Beispiel durchgerechnet

Klein zum Mitrechnen: Wörter statt Tokens, nur **M = 101** Zeilen (auch eine Primzahl) und eine **vereinfachte FNV-1a-Hashfunktion** — sie tut dasselbe: aus Text mach Zahl.

```
// 1 — kanonische Form
"Die Hauptstadt von Frankreich" -> "die hauptstadt von frankreich"

// 2 — Suffix-N-Gramme (n bis 3)
2-Gramm: ("von", "frankreich")
3-Gramm: ("hauptstadt", "von", "frankreich")

// 3 — jeder Kopf macht daraus eine Zeilennummer (mod 101)
2-Gramm -> Kopf 1: 67 Kopf 2: 44
3-Gramm -> Kopf 1: 5 Kopf 2: 73
E[2,1][67] = [ 0.12, -0.44, 0.91, ... ] // 1280 Zahlen

// 4+5 — gaten und addieren, in Schicht 2 und 15
h_neu = h_t + 0.8 * mittel(alle Koepfe)
```

Achte auf das, was *nicht* dasteht: kein $W \cdot x$, kein Softmax (Kapitel 050). Die Tabelle hat die Form [M, 1280], ein Lookup ist **Zeile auswählen und kopieren** — der Aufwand hängt nicht von der Tabellengröße ab.

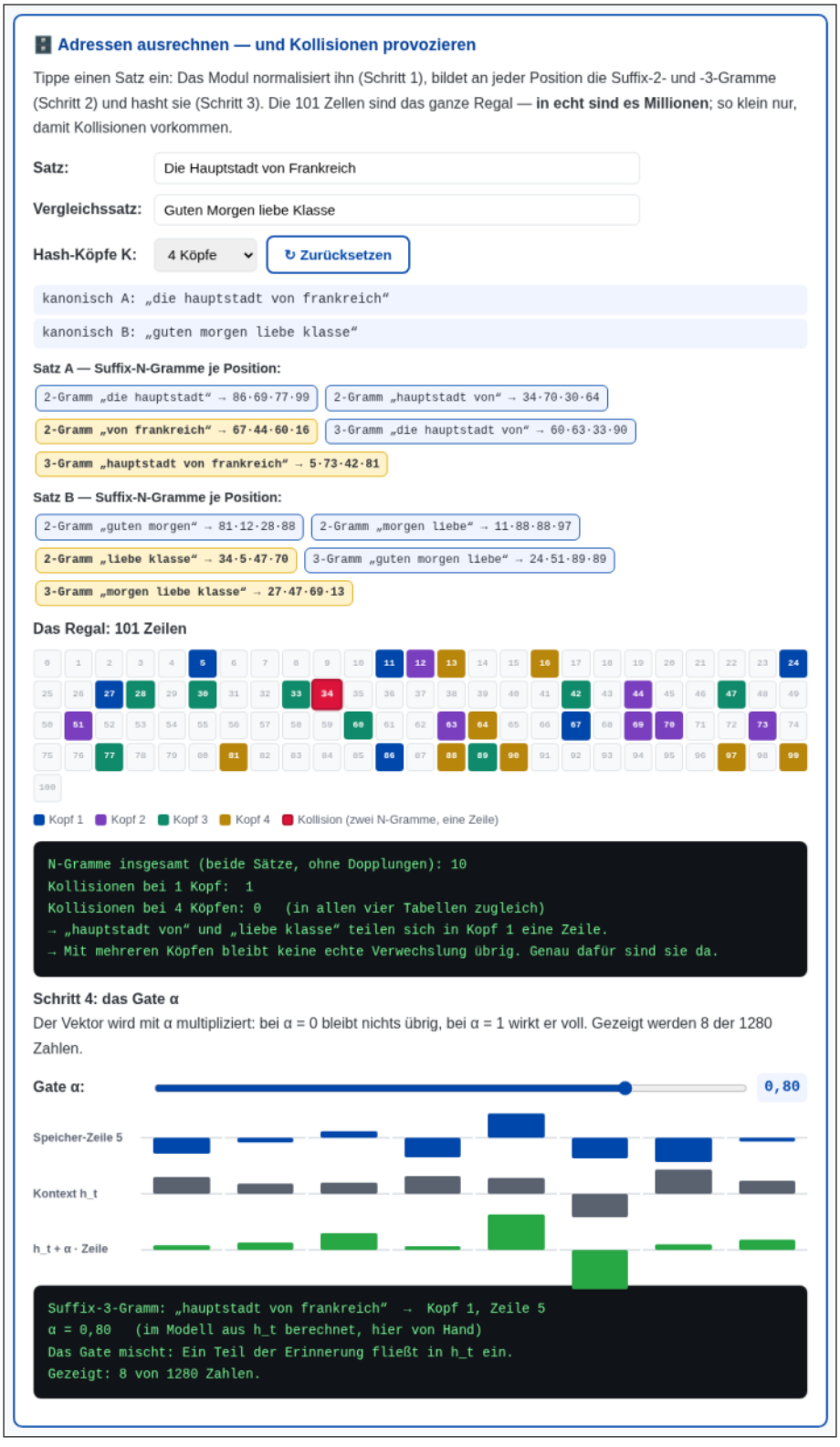
Der Haken: Kollisionen

Eine Hashfunktion presst unendlich viele N-Gramme auf endlich viele Zeilen. Irgendwann landen **zwei verschiedene N-Gramme in derselben Zeile** und sind nicht mehr unterscheidbar — eine **Kollision**. Engram löst das nicht, es *verdünnt* es: **K = 8 Hash-Köpfe**, jeder mit eigener Tabelle und Streuung. Was in Kopf 1 zusammenfällt, liegt in Kopf 2 fast sicher weit auseinander; erst wer in *allen acht* kollidiert, ist verloren. Die **Primzahlgröße** M streut zusätzlich gleichmäßig.

Zum Anfassen: die Hash-Tabelle

Am Gerät ausprobieren: herrki.de/335_ngram_speicher#abschnitt-4

Direkt zum Modul



Interaktives Modul – am Gerät ausprobieren

Warum das kein RAG ist

„Das Modell holt sich Wissen von außen“ — das kennst du aus Kapitel 330 als RAG. Beide klingen verwandt und sind es kaum: Es zählt, **was** geholt wird und **wann**.

	RAG (Kapitel 330)	Engram
Was wird geholt?	**Text** aus Dokumenten	**Trainierte Vektoren**
Wann gelernt?	gar nicht	im Training; Wissen *ist* Gewicht
Aktualisierbar?	ja, jederzeit	nur durch neues Training
Zitierbar?	ja	nein — ein Vektor hat keine Fundstelle
Wo liegt es?	Datenbank daneben, kommt in den Kontext	im Modell, kommt in den Hidden State
Kosten je Token	Suchlauf + Kontext	O(1), Adresse vorab bekannt

Merksatz: RAG ist das aufgeschlagene Buch neben dem Modell, Engram das Auswendiggelernte — und das belegt keine Denkkapazität mehr. Beides ergänzt sich: den *Wechselkurs* holst du per RAG, das *Grammatikmuster* „Sehr geehrte Damen und Herren“ hat das Modell besser im Regal.

Was es bringt — die Zahlen aus dem Paper

Der interessante Vergleich ist nicht „groß gegen klein“, sondern **gleiches Budget, andere Aufteilung**: **Engram-27B** gegen reines **MoE-27B**, iso-Parameter und iso-FLOPs.

	MoE-27B	Engram-27B
Parameter gesamt	26,7 Mrd.	26,7 Mrd.
aktiv je Token	3,8 Mrd.	3,8 Mrd.
Trainingstokens	262 Mrd.	262 Mrd.
Routed Experts	72	**55**
Speichertabelle	—	**5,7 Mrd. Parameter**

17 Experten abgebaut, die frei gewordenen Parameter ins Regal (ρ = 74,3 %; was ρ ist, kommt gleich). Das Ergebnis laut Paper in Punkten — Fachwissen (MMLU), dasselbe auf Chinesisch (CMMLU), Denkaufgaben (BBH), Sachfragen (ARC-C), Code, Mathe:

MMLU	CMMLU	BBH	ARC-C	HumanEval	MATH
+3,4	**+4,0**	**+5,0**	**+3,7**	**+3,0**	**+2,4**

Ein Wert fällt aus der Reihe: **Multi-Query Needle-in-a-Haystack** (mehrere „Nadeln“ in einem langen Text, alle auf einmal gesucht) — von **84,2 auf 97,0**.

Warum ausgerechnet der Langkontext-Test so springt, ist der schönste Beleg für die Grundthese: Muss die Attention die lokalen Muster nicht mehr selbst rekonstruieren, hat sie **Kapazität für den globalen Zusammenhang frei** — genau dafür ist sie gebaut (Kapitel 320). Das zweite Modell, **Engram-40B**, treibt die Tabelle bei gleichem aktivem Budget auf **18,5 Mrd. Parameter**: Der Verlust sinkt weiter und ist laut Paper **noch nicht gesättigt** — untertrainiert.

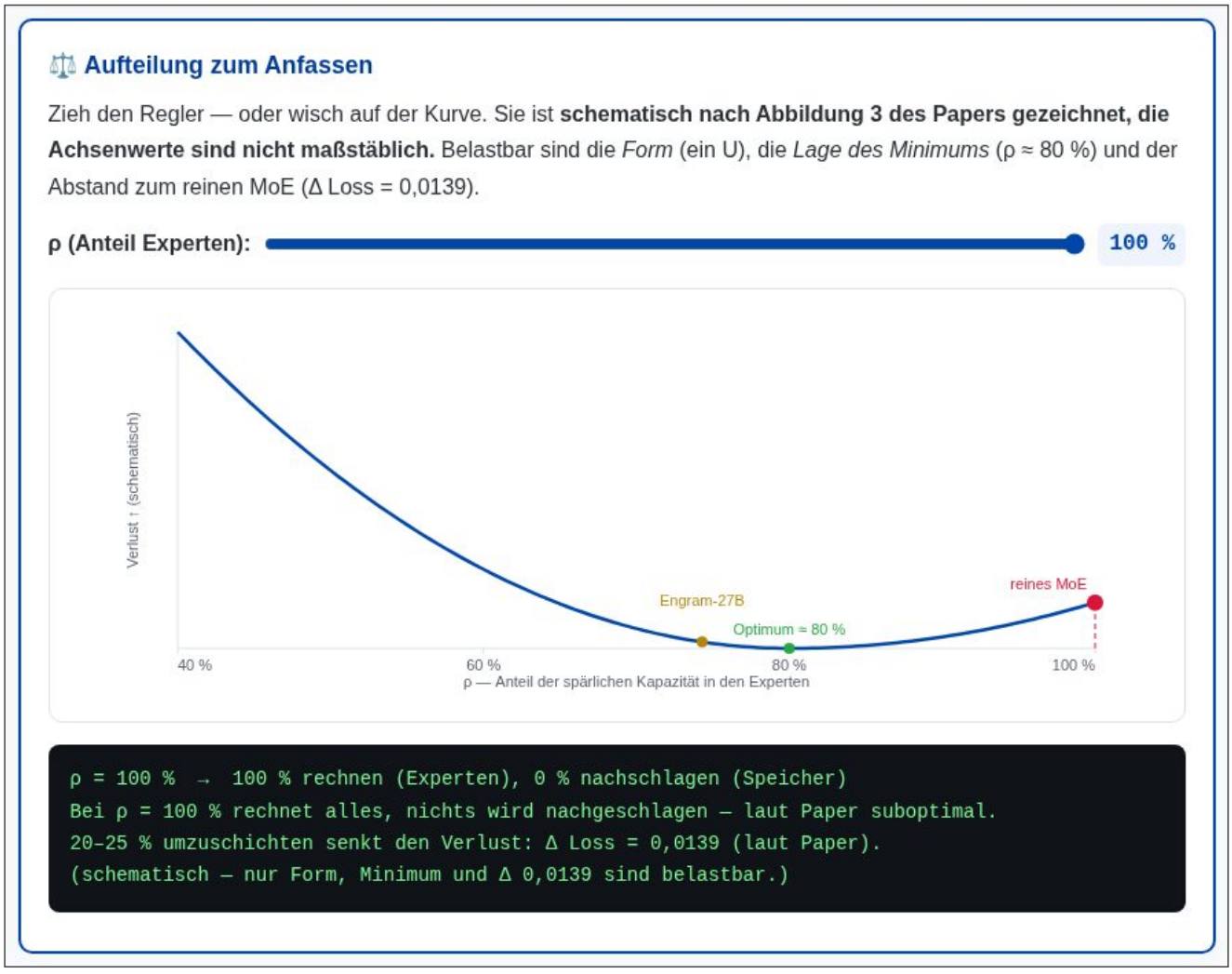
⚠ **Einordnung: Zwei bis fünf Punkte** — kein Sprung, keine neue Generation. Entscheidend ist: Sie entstehen bei **identischem Rechenbudget**. Nicht „mehr Rechnen bringt mehr“, sondern „dieselbe Rechnung, klüger aufgeteilt“.

Die U-Kurve: Wie viel Regal, wie viel Rechnen?

Warum nicht gleich alles nachschlagen? Weil ein Modell ohne Rechenkapazität ein Papagei wäre. Der Kennwert der Aufteilung heißt **ρ (rho)**: der **Anteil der spärlichen Kapazität in den MoE-Experten**. ρ = 100 % heißt „alles rechnet“, ρ = 60 % heißt „40 % des Budgets liegen im Regal“.

Am Gerät ausprobieren: herrki.de/335_ngram_speicher#abschnitt-7

Direkt zum Modul



Interaktives Modul – am Gerät ausprobieren

Zwei Befunde stecken darin. Erstens: **Reines MoE ist suboptimal** — schon **20 bis 25 %** des Budgets in den Speicher umzuschichten senkt den Trainingsverlust, das Minimum liegt bei **ρ ≈ 80 %** (**Δ Loss = 0,0139** zum reinen MoE). Zweitens bleibt die **Lage** des Optimums über Modellgrößen **stabil**. Vermessen mit einem Sweep von **2,58 × 10⁵ bis 1,0 × 10⁷** Speicherplätzen (bis etwa **13 Mrd. Parameter**) auf einem Backbone mit **100 Mrd. Trainingstokens**.

> **Ablation: Tokenizer-Kompression** (Schritt 1) und **Gating** (Schritt 4) sind die wichtigsten Bausteine — ohne sie bricht der Gewinn ein. Und **4-Gramme lohnen sich nicht**, jedenfalls

bei festem Budget: Sie sind so selten, dass ihre Zeilen kaum Training abbekommen, und verdünnen die häufigen 2- und 3-Gramme. Mehr Kontext ist hier *schlechter*.

Vertiefung: Warum die Tabelle in den Arbeitsspeicher darf

Jetzt der Teil, der die Idee praktisch macht — er hängt an Kapitel 390: Beim Antworten bremst nicht die Rechenleistung, sondern die **Bandbreite**. Die Brücke zwischen RAM und Grafikkarte (**PCIe 4.0 x16, ≈ 32 GB/s**) ist schmal — Gewichte darüber zu ziehen kostet *bei jedem Token* Zeit, und alles außerhalb des VRAM ist damit normalerweise erledigt.

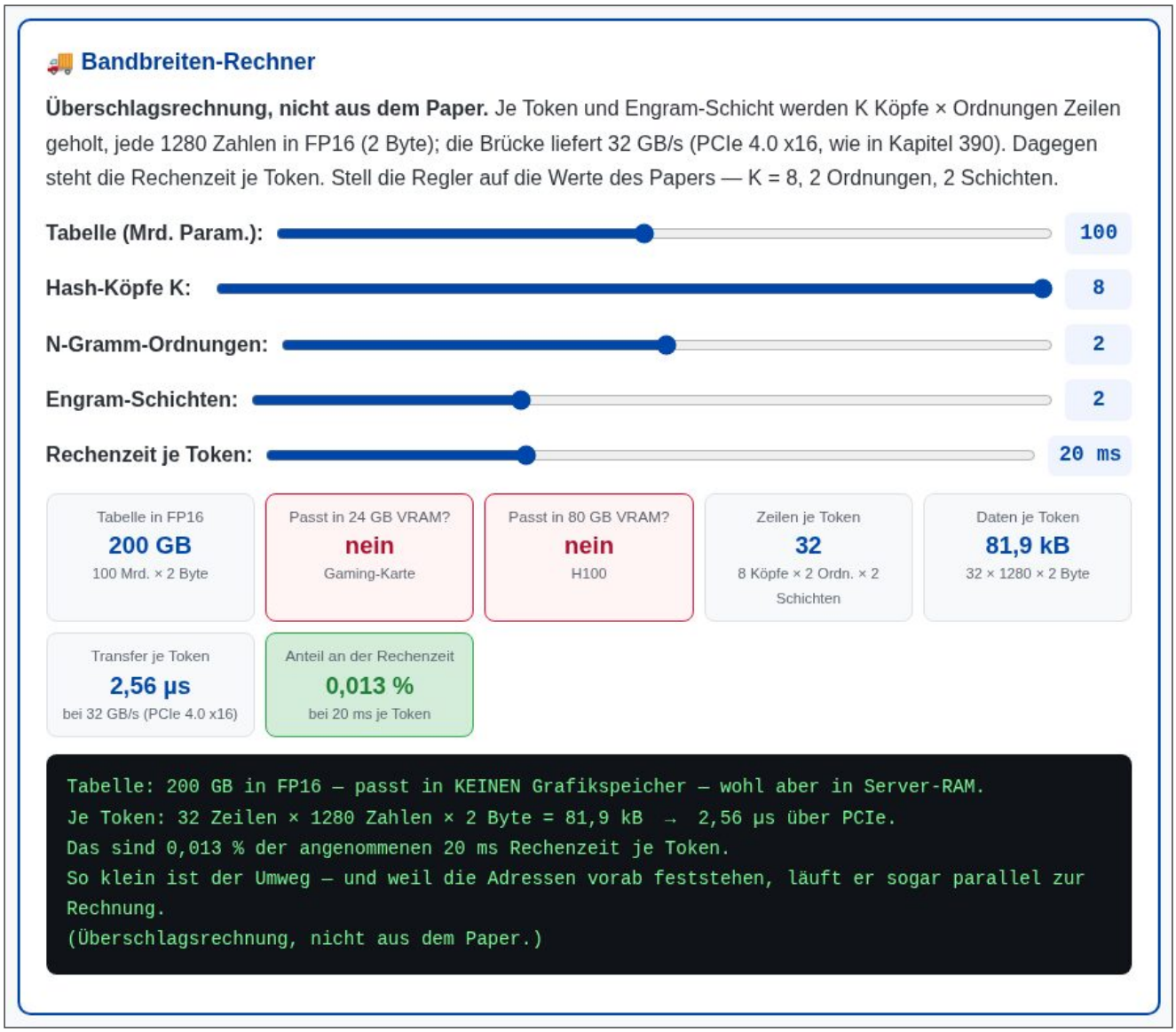
Der Unterschied: die Adresse steht vorher fest

Bei Engram ist eine Sache anders als bei allem anderen im Modell: Die Zeilennummern hängen **nur von den bereits bekannten Tokens** ab, nicht von einem Zwischenergebnis. Der Hash ist deterministisch — der Rechner weiß **schon vor dem Forward-Pass**, welche Zeilen er braucht. Damit wird aus dem Problem eine Fußnote: Der Host holt sie **asynchron** über PCIe, **während die vorhergehenden Schichten noch rechnen**. Genau deshalb sitzt der erste Engram-Block in **Schicht 2** statt in Schicht 1 (der zweite in Schicht 15): Schicht 1 ist der Vorsprung fürs Vorabholen. Ergebnis ist die erstaunlichste Zahl des Papers — eine **Tabelle mit 100 Mrd. Parametern im Host-Speicher, bei weniger als 3 % Durchsatzverlust**.

> **Und es geht billiger.** N-Gramme sind **Zipf-verteilt**: wenige („in der“) ständig, die meisten fast nie. Also lohnt ein **Mehrstufen-Cache** — häufige Zeilen im GPU-Speicher oder DRAM, seltene auf NVMe. Beim *Training* sind die Tabellen über die GPUs **geshardet** (All-to-All).

Am Gerät ausprobieren: herrki.de/335_ngram_speicher#abschnitt-8

Direkt zum Modul



Interaktives Modul – am Gerät ausprobieren

Zurück zur Anwendung: Wissen im billigen Speicher

- **Für lokale KI** (Kapitel 390): Dort gilt — was nicht ins VRAM passt, läuft nicht gut. Engram verschiebt einen Teil des Modells dorthin, wo Speicher *billig* ist: Faktenwissen im **Arbeitsspeicher**, im teuren Grafikspeicher nur noch das, was rechnet.
- **Für dich als Nutzer:** Modelle könnten Fakten **stabiler „wissen“** (eigene Adresse statt über die Schichten verstreut) und hätten die Denkkapazität frei — der Sprung im Langkontext-Test ist genau dieser Effekt.
- **Für Aktuelles bleibt RAG.** Ein Speichervektor ist ein Gewicht und veraltet wie jedes. Was sich ändert, gehört in die semantische Suche aus Kapitel 330.

⚠ **Realitätscheck:** Stand Januar/Juli 2026 ist Engram ein Paper mit offenem Code, ohne veröffentlichte Gewichte und nach unserem Wissensstand (September 2026) in keinem Produkt. Wenn dir jemand erzählt, sein Lieblings-Chatbot arbeite „mit Engram“ — nachfragen.

Was heißt das für dich?

- **Zwei Sorten Wissen.** Warum ein Modell bei Fakten patzt, beim Umformulieren aber glänzt: Nachschlagewissen und Rechenwissen — heute gleich teuer behandelt.
- **„Mehr Parameter“ ist nicht „mehr Rechnen“.** Modellgröße und Rechenlast sind zwei verschiedene Zahlen.
- **Achte auf „bei gleichem Rechenbudget“.** Der Satz unterscheidet die echte Architekturidee von schierer Größe.
- **Speicherhierarchie ist kein Nebenthema.** Ob etwas in HBM, DRAM oder auf der SSD liegen darf, entscheidet, was gebaut werden kann.

Zum Weiterlesen

- **Das Paper:** Cheng, Tian, Zeng, Dai u. a. (DeepSeek-AI), *„Conditional Memory via Scalable Lookup“*, 2026 — arxiv.org/abs/2601.07372
- **Der Code** (Apache-2.0, ohne Gewichte): github.com/deepseek-ai/Engram
- **Der Vorläufer:** Berges u. a., *„Memory Layers at Scale“*, 2024 — arxiv.org/abs/2412.09764
- **Woher MoE kommt:** Shazeer u. a., *„Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer“*, 2017 — arxiv.org/abs/1701.06538
- **N-Gramme als Embeddings:** Bojanowski u. a., *„Enriching Word Vectors with Subword Information“* (fastText), 2016 — arxiv.org/abs/1607.04606
- **Auf HerrKI:** 330 — Semantische Suche und RAG und 390 — KI im eigenen Haus

Kurzer Check – mit Lösung

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

Frage 1: Was schlägt Engram nach?

- Textstellen aus Dokumenten — so wie RAG in Kapitel 330.
- ✔ **Einen trainierten Vektor aus einer Tabelle, adressiert über die letzten bis zu drei Tokens.**
- Das nächste Token direkt — die Tabelle enthält Antworten.

Genau. Die letzten kanonischen Tokens werden gehasht, das Ergebnis ist eine Zeilennummer, und dort steht ein gelernter Vektor mit 1280 Zahlen — gegatet und auf den Hidden State addiert.

Frage 2: Warum darf die Tabelle im Arbeitsspeicher statt auf der Grafikkarte liegen?

- Weil sie klein genug ist, dass die Übertragung nicht ins Gewicht fällt.
- Weil der Arbeitsspeicher schneller ist als der Grafikspeicher.
- ✔ **Weil die Adressen vor dem Rechnen feststehen — die wenigen Zeilen je Token werden über PCIe vorab geholt, während frühere Schichten rechnen.**

Richtig: Der Hash hängt nur von bekannten Tokens ab, das Kopieren überlappt mit der Rechnung. Klein ist die Tabelle nicht (laut Paper 100 Mrd. Parameter bei unter 3 % Durchsatzverlust), und RAM ist langsamer als VRAM — Kapitel 390.

Frage 3: Wo liegt laut Paper das Optimum zwischen Rechnen und Nachschlagen?

- Bei 50 zu 50 — halbe Experten, halber Speicher.
- ✔ **Rund 80 % der spärlichen Kapazität rechnen, 20 % nachschlagen — reines MoE ist suboptimal.**
- Möglichst alles nachschlagen, das spart am meisten Rechenzeit.

Die Kurve ist ein U: Bei $\rho = 100$ % (reines MoE) liegt der Verlust um 0,0139 über dem Minimum bei $\rho \approx 80$ %; schichtet man mehr um, fehlt Rechenkapazität. Die Lage des Optimums bleibt über Modellgrößen stabil.

Das nimmst du mit

- **Zwei Achsen der Sparsamkeit.** MoE spart am Rechnen, Engram ergänzt das Nachschlagen („bedingtes Erinnern“).
- **Die letzten Wörter sind die Adresse.** Kanonische Tokens → Suffix-2/3-Gramm → Hash mod Primzahl → Zeile mit 1280 Zahlen. Acht Köpfe verdünnen die Kollisionen.
- **Das Gate entscheidet mit.** α aus dem Hidden State schaltet eine Erinnerung stumm, die nicht zum Kontext passt.
- **Es ist kein RAG.** RAG holt Text zur Laufzeit, zitierbar und aktualisierbar — Engram holt trainierte Gewichte.
- **Der Trick ist die vorab bekannte Adresse.** Deshalb darf ein 100-Mrd.-Regal im billigen Arbeitsspeicher stehen — für unter 3 % Durchsatz.

Weiter geht’s mit einer anderen Sorte Modell: In „Diffusion“ siehst du Bild-KIs, die ein ganzes Bild aus Rauschen herausschälen statt Wort für Wort zu erzeugen. Zusammen mit Kapitel 330 kennst du damit beide Wege, wie Wissen in ein Modell kommt: von außen geholt oder innen einsortiert.

So zitierst du dieses Kapitel

HerrKI: Nachschlagen statt rechnen: N-Gramm-Speicher im Modell. herrki.de/335_ngram_speicher, ein privates Hobbyprojekt von Terragon.de, abgerufen am 09.09.2026.