KI im eigenen Haus: lokale Modelle, VRAM, CPU gegen GPU

Kapitel online

Warum ein 8-Milliarden-Modell auf deine Grafikkarte passt (oder nicht), was Quantisierung mit den Gewichts-Tensoren macht, warum die GPU beim Lesen deines Prompts hundertmal schneller ist als die CPU, beim Antworten aber nur zehnmal — und wie Apple Silicon und DGX-Systeme den Speicher teilen. Mit Quantisierung zum Anfassen, VRAM-Rechner und CPU-gegen-GPU-Wettrennen.

Aktuell · Kapitel 40 von 50 · ca. 30 Min. · 300 XP

Einfach gesagt: Ein KI-Modell ist nur eine lange Zahlenliste — du rechnest aus, ob sie in den Speicher deiner Grafikkarte passt und wie schnell dein Rechner sie liest.

Du hast einen Arztbrief auf dem Tisch. Oder die Steuererklärung. Eine KI könnte dir erklären, was da steht — nur müsstest du die Datei dafür hochladen. Auf einen fremden Rechner. Es geht auch anders: **Das Modell kommt zu dir.** Es liegt als Datei auf deiner Festplatte, rechnet auf deiner Hardware, läuft ohne Empfang und kostet nichts pro Frage.

Zwei Fragen treiben dieses Kapitel an: **„Passt so ein Modell auf meinen Rechner?“** und **„Warum reden alle über VRAM und Grafikkarten?“** Am Ende rechnest du beides selbst aus.

HerrKI-Tipp: Ein Satz für dieses Kapitel — **„Rechnen ist billig, Speicher-Lesen ist teuer.“** Fast alles Merkwürdige an lokaler KI folgt daraus.

Was du hier lernst

- Wie groß ein Modell ist: **Parameter × Bits ÷ 8**
- Was **Quantisierung** auf Tensor-Ebene macht — und warum das erstaunlich gut geht
- Warum die GPU beim Lesen deines Prompts hundertmal schneller ist, beim Antworten nur zehnmal — Stichwort **Speicherbandbreite**
- Drei **Speicher-Architekturen**: Gaming-PC, Apple Silicon / DGX Spark, Rechenzentrum
- Den **KV-Cache** — und die ehrliche Abwägung Datenschutz gegen Leistung

Für den Unterricht

Einstieg (6 Min): Kapitel gemeinsam lesen, Alltagsbeispiel aufgreifen.

Erkunden am Gerät (18 Min): die interaktiven Module auf herrki.de/390_lokale_ki

ausprobieren.

Sichern mit Quiz (6 Min): „Kurzer Check" besprechen, „Das nimmst du mit" festhalten.

Was ein Modell auf der Festplatte ist: Milliarden Zahlen

Ein Sprachmodell ist eine sehr lange Liste von Zahlen. Diese Zahlen heißen **Parameter** oder **Gewichte** — genau die Werte, die beim Training eingestellt (Kapitel 110) und beim Fine-Tuning verschoben werden (Kapitel 370). „8B“ heißt: **8 Milliarden Zahlen.** Damit ist die Dateigröße keine Geschmackssache mehr, sondern eine Multiplikation — üblich sind **16 Bit je Zahl** (Format FP16, also 2 Byte):

```
// Die einzige Formel, die du dir merken musst:
Groesse ≈ Parameter × Bits ÷ 8 (Ergebnis in Byte)

8 000 000 000 × 16 ÷ 8 = 16 000 000 000 Byte = 16 GB
```

Sechzehn Gigabyte — daran scheitern die meisten Grafikkarten (RTX 4060: 8 GB). Deshalb gleich die Quantisierung; vorher die Größenordnungen:

Modellgröße	FP16 (16 Bit)	Q8 (8 Bit)	Q4 (4 Bit)
8 Milliarden	16 GB	8 GB	4 GB
14 Milliarden	28 GB	14 GB	7 GB
32 Milliarden	64 GB	32 GB	16 GB
70 Milliarden	140 GB	70 GB	35 GB

Reine Rechenwerte: Echte Dateien sind **ca. 5–8 % größer**, weil Skalierungsfaktoren und die Einbettungstabellen (Token → Vektor) mitgespeichert werden. Real bei Llama-3-8B: FP16 ca. **16,1 GB**, Q8_0 ca. **8,5 GB**, Q4_K_M ca. **4,9 GB**.

Runter zum Tensor: Quantisierung — aus 16 Bit werden 4

Quantisierung (von „Quantum“, also Stufe) heißt: Jede Zahl wird mit weniger Bits gespeichert. Der wichtigste Punkt wird fast immer falsch erzählt — es wird *nichts weggeworfen*, nur *gröber gerastert*.

Das Rezept: 32 Gewichte teilen sich einen Maßstab

Die Gewichts-Tensoren werden in **Blöcken zu je 32 Gewichten** bearbeitet. Je Block kommt ein Wert dazu: der **Skalierungsfaktor** (englisch *scale*) — der Maßstab, mit dem dieser Block gelesen wird, wie auf einer Landkarte „1 cm entspricht 500 m“.

```
// Ein Block von 32 Gewichten, symmetrisch auf 4 Bit (Werte −7 … +7)
scale = max|w| ÷ 7 ← der groesste Betrag im Block bestimmt den Massstab
q = round(w ÷ scale) ← gespeichert wird nur diese kleine ganze Zahl
w′ = q × scale ← beim Rechnen zurueckgerechnet, in Echtzeit
```

`q` passt in 4 Bit, `scale` gibt es nur einmal je 32 Gewichte. *Ehrlichkeitshinweis:* vereinfachte Darstellung — llama.cpp nutzt −8 bis 7 und weitere Tricks, fürs Verständnis ändert das nichts. Der Maßstab kostet selbst Platz, den man mitzählen muss:

```
// Q4_0 — 4 Bit je Gewicht, ein 16-Bit-Massstab je 32er-Block
32 × 4 Bit + 16 Bit = 144 Bit = 18 Byte je 32 Gewichte → 4,5 Bit je Gewicht

// Q8_0 — 8 Bit je Gewicht, gleicher Aufschlag
32 × 8 Bit + 16 Bit = 272 Bit = 34 Byte je 32 Gewichte → 8,5 Bit je Gewicht

// Q4_K_M — feinere Skalierung fuer die wichtigen Bloecke
                                                    ≈ 4,85 Bit je Gewicht
```

`K` steht für die feinere Skalierungs-Methode, `M` für „medium“: Die empfindlichen Schichten bekommen mehr Bits. Diese 0,35 Bit Aufpreis machen Q4_K_M zum beliebtesten Format der Praxis.

Warum Sprachmodelle das vertragen

Alles hängt an Kapitel 060: Was eine Schicht ausgibt, ist eine **gewichtete Summe** über tausende Gewichte. Liegt jedes einzelne ein winziges bisschen daneben, mal zu hoch, mal zu tief, heben sich diese Rundungsfehler in der Summe großteils gegenseitig auf — das Ergebnis wackelt, es kippt nicht. Derselbe Gedanke wie „alles auf eine gemeinsame Skala“ aus Kapitel 135: Es zählen die Verhältnisse, nicht die letzte Nachkommastelle.

> ⚠ **Wo es kippt.** Bei Q3 und erst recht Q2 wird das Raster so grob, dass die Texte merkbar schlechter werden. Messen lässt sich das an der **Perplexität** — grob: wie überrascht das Modell vom tatsächlich folgenden Wort ist. Je überraschter, desto schlechter das

Sprachgefühl. Bei Q4 steigt sie kaum, bei Q2 deutlich. **Q4 ist der Punkt, an dem der Handel fair ist.**

Am Gerät ausprobieren: herrki.de/390_lokale_ki#abschnitt-3

Direkt zum Modul

🔬 Quantisierung zum Anfassen

Acht Gewichte, wie sie in einem Tensor stehen könnten. Schieb den Regler: Bei 8 Bit siehst du kaum einen Unterschied, bei 2 Bit wird es brutal.

Bits je Gewicht: 4

🎲 Andere Zufallsgewichte | ↻ Beispiel zurücksetzen

`0,84` `-0,14` `0,00` `-0,56` `0,28` `-0,98` `0,42` `-0,14`

#	Original w	ganzzahlig q	zurück w'	Fehler |w − w'|
1	0,82	6	0,840	0,020
2	-0,15	-1	-0,140	0,010
3	0,03	0	0,000	0,030
4	-0,61	-4	-0,560	0,050
5	0,27	2	0,280	0,010
6	-0,98	-7	-0,980	0,000
7	0,44	3	0,420	0,020
8	**-0,07**	**-1**	**-0,140**	**0,070**

FP16 — Original `16,0 Bit`

4 Bit + Maßstab `4,5 Bit`

```
Skalierungsfaktor  scale = max|w| ÷ 7 = 0,1400
Wertebereich von q: −7 … +7   (4 Bit)
mittlerer Fehler  ≈ 0,0263   ·   größter Fehler ≈ 0,0700
Speicher je Gewicht: 4,5 Bit  →  ein 8-Mrd.-Modell wäre ≈ 4,5 GB groß
```

Die Balken zählen den 16-Bit-Maßstab je 32er-Block mit — deshalb bei 4 Bit nicht „4,0“, sondern **4,5 Bit** je Gewicht: genau die Zahl, die oben als `Q4_0` herauskam.

Interaktives Modul – am Gerät ausprobieren

Warum die Grafikkarte? CPU und GPU im Vergleich

CPU — der Alleskönner

- **Wenige starke Kerne** (typisch 8–16), jeder kann fast alles.
- Große Zwischenspeicher, sehr gut im Umschalten zwischen Aufgaben.
- Rechnet mit **SIMD** („ein Befehl, viele Daten“) 8–16 Zahlen pro Befehl.
- Stark bei: *viele verschiedene Dinge nacheinander.*

GPU — der Gleichschritt-Spezialist

- **Tausende einfache Recheneinheiten** (RTX 4090: 16 384 Stück).
- Arbeiten in Gruppen zu 32 im Gleichschritt — alle machen denselben Befehl.
- Dazu **Tensor-Kerne**, die kleine Matrizen in einem Schritt multiplizieren.
- Stark bei: *eine Sache tausendfach gleichzeitig.*

Und ein Sprachmodell tut die ganze Zeit genau eines: **Matrixmultiplikationen** (Kapitel 060). Die bestehen aus tausenden unabhängigen „multiplizieren und addieren“-Schritten, die alle gleichzeitig laufen dürfen. Exakt die Aufgabenform, für die eine GPU gebaut ist.

Die Wendung, die kaum jemand kennt

> **Beim Erzeugen von Text, Token für Token, ist nicht das Rechnen der Engpass, sondern das Herbeischaffen der Gewichte aus dem Speicher.**
>
> Für jedes neue Token muss **jede Gewichtsmatrix einmal komplett gelesen werden** — bei einem 4,9-GB-Modell also 4,9 GB durch die Leitung, für ein einziges Wort. Gerechnet wird dabei nur „Matrix mal Vektor“ je Schicht, und das ist für eine GPU eine Lachnummer. Die Recheneinheiten warten auf Nachschub.

```
// Daumenregel fuer das Tempo jeder beliebigen Hardware:
Tokens pro Sekunde ≈ 0,7 × Speicherbandbreite ÷ Modellgroesse

// Die 0,7 sind der Wirkungsgrad – kein Speicher liefert seine Katalogzahl wirklich ab.
```

Die Speicherbandbreite steht auf keinem Werbeplakat — und entscheidet alles *(alle Werte ca., Herstellerangaben)*:

Speicher	Bandbreite (ca.)	8B in Q4 (≈ 4,9 GB)
PC-Arbeitsspeicher DDR5, zwei Kanäle	≈ 90 GB/s	≈ 13 Tokens/s
NVIDIA DGX Spark	≈ 273 GB/s	≈ 39 Tokens/s
Apple M4 Max	≈ 546 GB/s	≈ 78 Tokens/s
Apple M3 Ultra	≈ 819 GB/s	—
NVIDIA RTX 4090	≈ 1 008 GB/s	≈ 145 Tokens/s
NVIDIA H100	≈ 3 350 GB/s	—

Gerundete Schätzungen nach der Formel oben; in der Praxis etwas weniger. Zum Einordnen: Flüssiges Lesen sind grob 5 bis 10 Wörter pro Sekunde — die 13 Tokens/s der reinen CPU sind also nicht unbenutzbar, sie fühlen sich nur an wie jemand, der beim Tippen nachdenkt.

Zwei Phasen, zwei völlig verschiedene Engpässe

1 Prefill — dein Prompt wird gelesen

Alle Tokens deiner Eingabe werden **auf einmal** verarbeitet: **Matrix mal Matrix**. Jetzt haben die Recheneinheiten wirklich etwas zu tun.

→ rechenbegrenzt. Die GPU ist **50- bis 100-mal schneller** als die CPU.

2 Decode — die Antwort entsteht

Ein Token nach dem anderen, für jedes müssen alle Gewichte durch die Leitung: **Matrix mal Vektor** — wenig Rechnung, viel Transport.

→ bandbreitenbegrenzt. Die GPU ist nur noch **≈ 11-mal** schneller, denn 1 008 ÷ 90 ≈ 11.

Für die Prefill-Seite zählt die zweite Kennzahl: die **Rechenleistung** in TFLOPS (Billionen Rechenoperationen je Sekunde) — *grob, ca., Herstellerangaben*:

Hardware	FP16 (ca.)	Hardware	FP16 (ca.)
CPU, 16 Kerne	≈ 1,5 TFLOPS	RTX 4090	≈ 165 TFLOPS
Apple M4 Max	≈ 35 TFLOPS	RTX 5090	≈ 210 TFLOPS
RTX 4060	≈ 60 TFLOPS	NVIDIA DGX Spark	≈ 125 TFLOPS
RTX 4070	≈ 120 TFLOPS	H100	≈ 990 TFLOPS

```
// Wie lange dauert das Lesen eines langen Prompts?
Rechenaufwand ≈ 2 × Parameter × Tokens

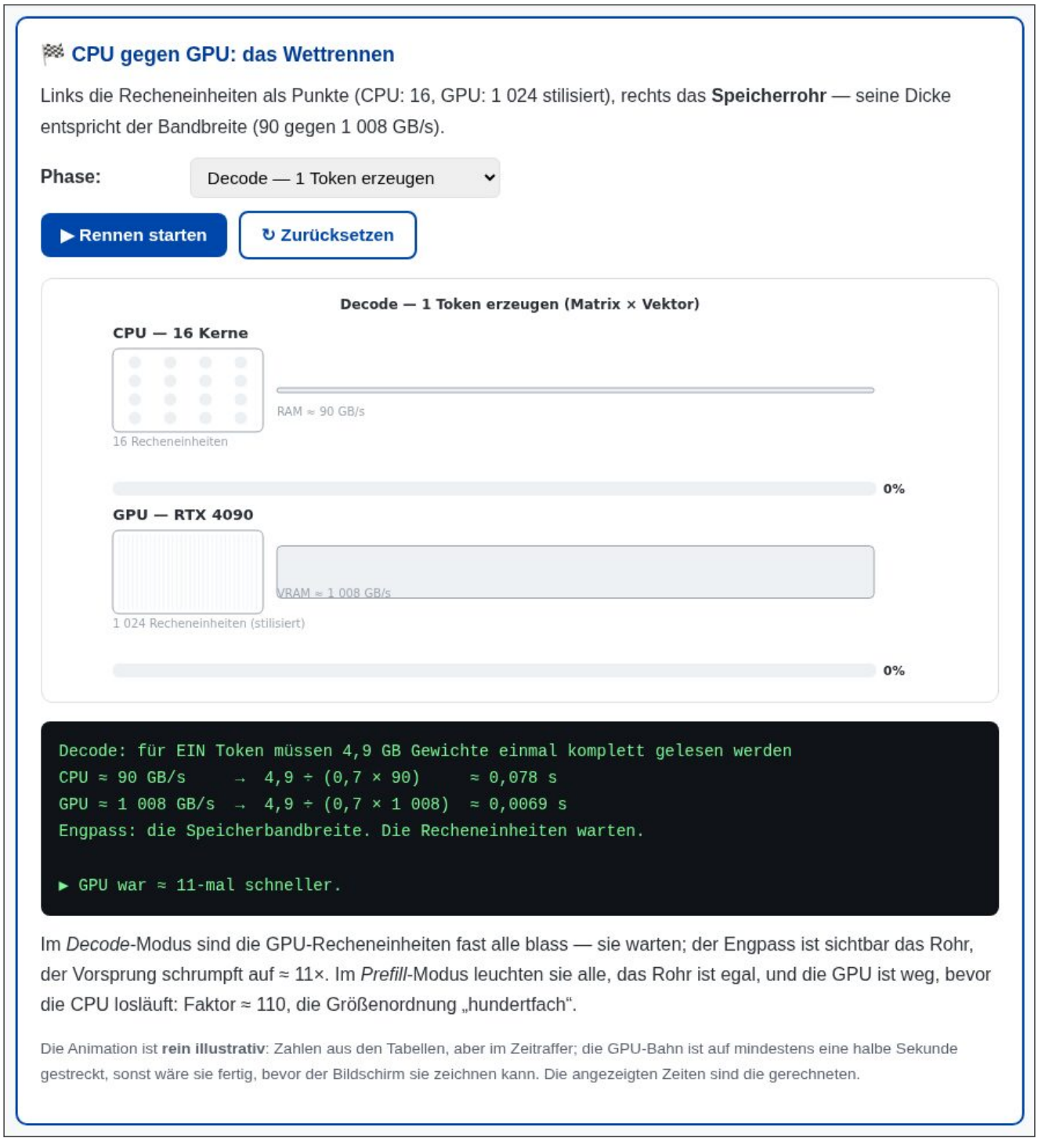
// 8-Mrd.-Modell, 2 000 Tokens Prompt (ca. drei Seiten Text):
2 × 8 000 000 000 × 2 000 = 32 TFLOP

RTX 4090: 32 ÷ 165 ≈ 0,2 Sekunden
CPU: 32 ÷ 1,5 ≈ 21 Sekunden ← das spuert man
```

Das erklärt „die KI denkt erst ewig nach und schreibt dann normal“: Auf einer CPU dauert allein das *Lesen* 21 Sekunden. **Und die NPUs?** Die „Neural Engine“ in Handys und Notebooks ist für kleine Modelle mit sehr wenig Strom optimiert; für große Sprachmodelle spielt sie heute eine Nebenrolle.

Am Gerät ausprobieren: herrki.de/390_lokale_ki#abschnitt-4

Direkt zum Modul



Interaktives Modul – am Gerät ausprobieren

Wo liegt das Modell? Drei Speicher-Architekturen

„Passt das Modell?“ heißt eigentlich: *Passt es in den Speicher, an dem der Rechenchip hängt?* Darauf gibt es drei sehr verschiedene Antworten.

(a) Gaming-PC

CPU 64 GB RAM

↕

PCIe 4.0 x16
≈ 32 GB/s

↕

GPU 24 GB VRAM

Zwei **getrennte** Speicher, verbunden durch eine schmale Brücke. Das Modell muss ins VRAM passen — sonst hilft nur **Offloading**: Ein Teil der Schichten bleibt im CPU-RAM und muss *für jedes Token* durch die 32-GB/s-Engstelle. Deutlich langsamer.

(b) Unified Memory

CPU + **GPU**

↓

EIN gemeinsamer Speicherpool
bis 512 GB

Apple Silicon und NVIDIA DGX Spark teilen sich **einen** Speicher, nichts wird mehr hinüberkopiert *(ca., Herstellerangaben)*:

- M4 Max: bis **128 GB** bei ≈ 546 GB/s
- M3 Ultra: bis **512 GB** bei ≈ 819 GB/s
- DGX Spark: **128 GB** bei ≈ 273 GB/s

Vorteil: 70-Milliarden-Modelle passen überhaupt erst hinein. **Nachteil:** Bandbreite unter GDDR6X/HBM, also weniger Tokens/s als eine 4090 — bei einem Modell, das dort gar nicht

startet, trotzdem der Gewinn.

(c) Rechenzentrum

DGX H100 8 × 80 GB HBM3

↓

je GPU ≈ 3 350 GB/s
NVLink ≈ 900 GB/s je GPU

Das Modell passt in *keine* einzelne GPU, also wird es verteilt: Bei der **Tensor-Parallelität** rechnet jede GPU einen Teil *jeder* Matrix, nach jeder Schicht werden die Teilergebnisse zusammengeführt. Deshalb die extrem schnellen Verbindungen — ein normales Netzwerk wäre viel zu langsam. Genau der Unterschied zum „Freundeskreis-Cluster“ aus Kapitel 380.

Vertiefung: Der KV-Cache — der Speicherfresser, den die Faustformel vergisst

Der Teil, der in fast jeder VRAM-Rechnung im Internet fehlt. Erinnere dich an Kapitel 320: Attention arbeitet je Token mit Query, Key und Value. Beim 500. Token schaut das Modell auf alle 499 davor — es müsste deren Schlüssel und Werte jedes Mal neu ausrechnen. Macht es nicht: Es **merkt sie sich**, für jedes Token in jeder Schicht. Dieser Merkzettel heißt **KV-Cache**.

```
// Wie viel Platz braucht der Merkzettel je Token?
Byte je Token = 2 × Schichten × KV-Koepfe × Kopfdimension × Byte je Zahl
                ↑ einmal Key, einmal Value

// Llama-3-8B: 32 Schichten, 8 KV-Koepfe, Kopfdimension 128, FP16 (2 Byte)
2 × 32 × 8 × 128 × 2 = 131 072 Byte = 128 KB je Token

    8 000 Tokens Kontext → ≈ 1 GB
  128 000 Tokens Kontext → ≈ 16 GB ← so gross wie das Modell selbst!
```

Lies die letzte Zeile noch einmal: Bei voller Kontextlänge braucht der *Merkzettel* so viel Platz wie das ganze Modell. „Großer Kontext“ ist nichts, was man einfach so anschaltet.

Warum moderne Modelle das entschärfen: Grouped-Query-Attention

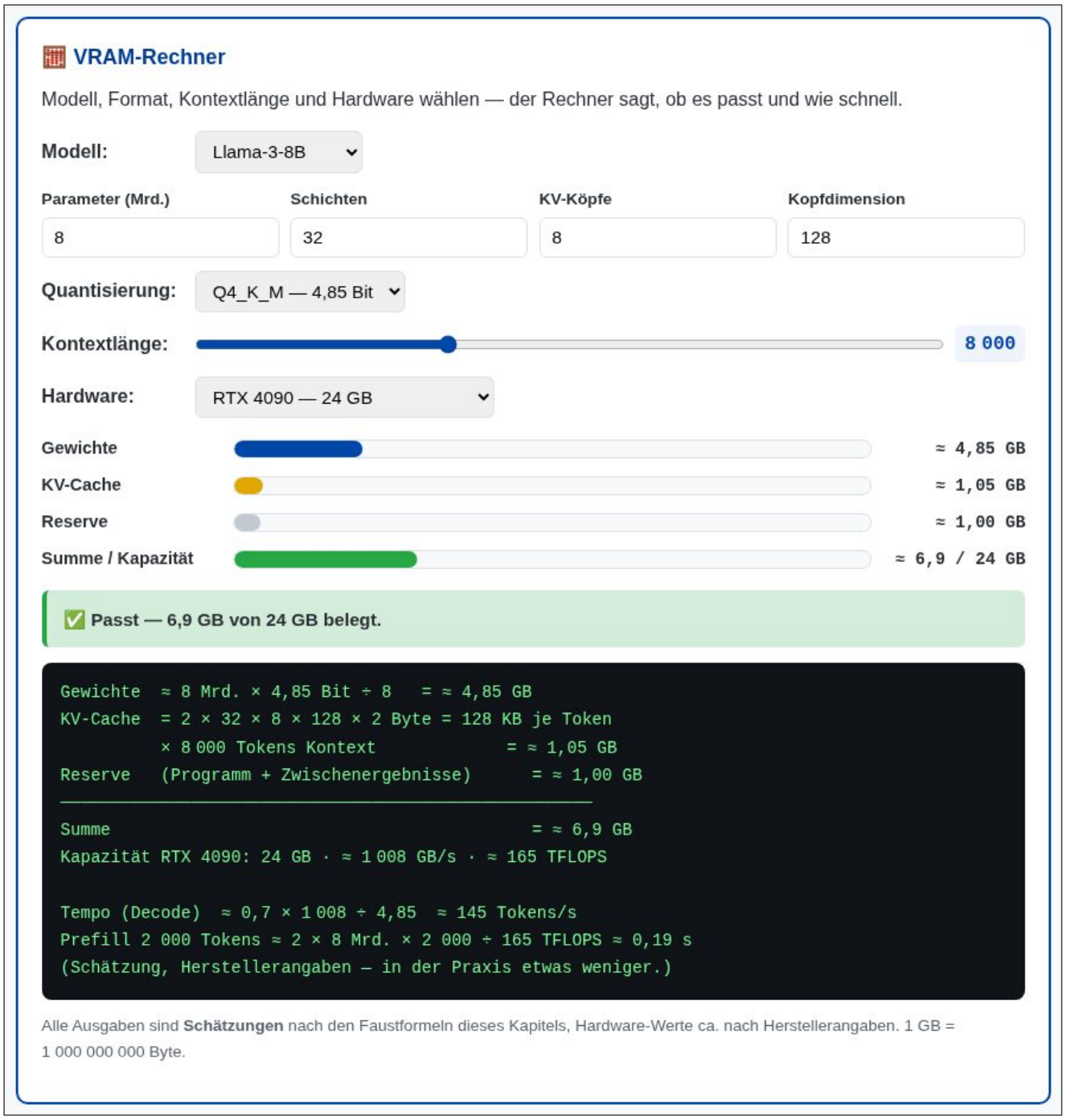
In der Formel stehen **KV-Köpfe**, nicht Köpfe: Ältere Modelle wie Llama-2-7B hatten für jeden der 32 Attention-Köpfe einen eigenen Schlüssel- und Wert-Satz, moderne lassen sich mehrere Köpfe einen Satz *teilen* — **Grouped-Query-Attention** (GQA):

Modell	Schichten	KV-Köpfe	Kopfdim.	KV je Token
Llama-2-7B (ohne GQA)	32	32	128	512 KB
Llama-3-8B (mit GQA)	32	8	128	128 KB
Llama-3.1-70B (mit GQA)	80	8	128	320 KB

Beim 70B-Modell sind **32 000 Tokens Kontext also ≈ 10 GB** — zusätzlich zu den Gewichten. Damit ist „Parameter × Bits ÷ 8“ allein zu optimistisch. Die ehrliche Rechnung lautet: **Gewichte + KV-Cache + ≈ 1 GB Reserve** für Programm und Zwischenergebnisse. Genau das rechnet dir der nächste Kasten aus.

Am Gerät ausprobieren: herrki.de/390_lokale_ki#abschnitt-6

Direkt zum Modul



Interaktives Modul – am Gerät ausprobieren

Zurück zur Anwendung: Datenschutz gegen Leistung — ehrlich abgewogen

Für lokal	Gegen lokal
Deine Daten verlassen das Haus nicht — dieselbe Idee wie die Textsuche in Kapitel 470, die komplett im Browser läuft.	Ein 8B-Modell in Q4 ist kein Spitzenmodell. Der Abstand zu den großen Diensten ist real und bei schwierigen Aufgaben spürbar.
Offline nutzbar, und keine Kosten je Token — du kannst experimentieren, so viel du willst.	Anschaffung und Strom: Eine RTX 4090 zieht unter Last ≈ 450 Watt.
Eigenes Feintuning möglich — LoRA aus Kapitel 370 läuft auf einer einzelnen GPU.	Pflege: Treiber, Updates, kaputte Formate.
Reproduzierbar: dieselbe Datei gibt in zwei Jahren dieselben Antworten, kein stilles Modell-Update.	Du bist deine eigene Sicherheitsabteilung. Niemand patcht das für dich.

Mein Vorschlag ist deshalb ein Hybrid: lokal für alles, was oft passiert und Daten anfasst — Suche, Einbettungen, Zusammenfassungen, erste Entwürfe. Das große Modell im Netz für die wenigen wirklich harten Denkaufgaben, bei denen du bewusst entscheidest, welche Daten mitgehen.

Die Werkzeuge — je ein Satz

- **llama.cpp** — die Engine unter der Haube vieler Programme; von hier stammen die GGUF-Dateien und die Q4/Q8-Formate.
- **Ollama** und **LM Studio** — bequeme Oberflächen darauf, die dir Herunterladen und Starten abnehmen.
- **vLLM** — Server mit **Batching** für viele Nutzer: Mehrere Anfragen zusammen machen aus „Matrix mal Vektor“ wieder „Matrix mal Matrix“, deshalb lohnt sich dort die Rechenleistung.
- **MLX** — Apples eigenes Framework, gebaut für den gemeinsamen Speicher der M-Chips.

Dateinamen wie `Q4_K_M` kannst du jetzt lesen: 4 Bit je Gewicht, K-Skalierung, mittlere Stufe, macht ≈ 4,85 Bit — bei 8 Milliarden Parametern also ≈ 4,9 GB.

Was heißt das für dich?

1. **Sind die Daten sensibel?** Wenn ja, ist lokal nicht die bequemste, aber die einzige Antwort, bei der du nichts erklären musst.
2. **Was passt in deinen Speicher?** Trag deine Hardware und die gewünschte Kontextlänge oben in den Rechner ein. Die Antwort ist eine Zahl, kein Gefühl.
3. **Die Bandbreite bestimmt das Tempo** — nicht der Preis. Bandbreite durch Modellgröße teilen, fertig.
4. **Denk hybrid.** Die Frage ist selten „lokal oder Cloud?“, sondern „welcher Teil meiner Arbeit gehört wohin?“

Zum Weiterlesen

- **llama.cpp** — die Engine hinter fast allem: github.com/ggml-org/llama.cpp
- **GPTQ** (Quantisierung großer Modelle): arxiv.org/abs/2210.17323
- **LLM.int8()** (warum manche Gewichte mehr Bits brauchen): arxiv.org/abs/2208.07339
- **Horace He, „Making Deep Learning Go Brrrr From First Principles“** — Speicher- gegen Rechenbegrenzung: horace.io/brrr_intro.html
- **NVIDIA DGX Spark, Systemübersicht:** docs.nvidia.com/dgx/dgx-spark/system-overview.html
- **LMSYS, DGX-Spark-Test:** lmsys.org/blog/2025-10-13-nvidia-dgx-spark
- **Auf HerrKI:** 010 — Was sind Tensoren? · 135 — Daten vorbereiten · 380 — KI-Agenten

Kurzer Check – mit Lösung

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

Frage 1: Was macht Quantisierung mit den Gewichten?

- Sie löscht die kleinsten Gewichte, damit weniger Zahlen übrig bleiben.
- **✓ Sie speichert sie mit weniger Bits — blockweise mit einem gemeinsamen Skalierungsfaktor; beim Rechnen wird zurückgerechnet.**
- Sie trainiert das Modell mit weniger Parametern neu, bis es kleiner ist.

Genau: q = round(w ÷ scale) beim Speichern, w′ = q × scale beim Rechnen. Je 32 Gewichte gibt es einen Maßstab — deshalb kostet Q4_0 nicht 4,0, sondern 4,5 Bit je Gewicht.

Frage 2: Warum entscheidet beim Antworten die Speicherbandbreite und nicht die Rechenleistung?

- Weil einzelne Tokens so große Datenmengen sind.
- Weil die GPU beim Antworten pausiert und nur die CPU weiterläuft.
- **✓ Weil für jedes einzelne Token alle Gewichte einmal komplett aus dem Speicher gelesen werden müssen.**

Richtig — daher „Tokens/s ≈ 0,7 × Bandbreite ÷ Modellgröße“. Gerechnet wird nur Matrix mal Vektor, die Recheneinheiten warten. Beim Prefill ist es umgekehrt: viele Tokens auf einmal, Matrix mal Matrix — da zählt die Rechenleistung.

Frage 3: Was ist der Vorteil von Unified Memory gegenüber einer Gaming-Grafikkarte?

- **✓ CPU und GPU teilen sich einen großen Speicherpool — das Modell muss nicht in ein getrenntes VRAM passen.**
- Die Speicherbandbreite ist höher als bei einer Grafikkarte.
- Man braucht dort keine Quantisierung mehr.

Genau: keine PCIe-Engstelle, kein Umkopieren — und mit 128 oder 512 GB passen Modelle hinein, die auf einer 24-GB-Karte gar nicht starten. Die Bandbreite ist dabei allerdings niedriger (≈ 546 gegen ≈ 1 008 GB/s): Tempo gegen Größe getauscht.

Das nimmst du mit

- **Größe ≈ Parameter × Bits ÷ 8.** 8 Milliarden Zahlen sind in FP16 16 GB, in Q4 ≈ 4,9 GB.
- **Quantisierung rastert, sie löscht nicht.** 32 Gewichte teilen sich einen Maßstab; die Rundungsfehler mitteln sich in der gewichteten Summe heraus.
- **Beim Antworten zählt die Bandbreite.** Alle Gewichte je Token einmal durch die Leitung — ≈ 145 Tokens/s auf einer 4090, ≈ 13 auf der CPU.
- **Beim Lesen zählt die Rechenleistung.** Prefill ist Matrix mal Matrix: 0,2 Sekunden auf der GPU gegen 21 Sekunden auf der CPU.
- **Der KV-Cache gehört in die Rechnung.** 128 KB je Token summieren sich bei vollem Kontext auf die Größe des Modells selbst — plus ≈ 1 GB Reserve.

Weiter geht's: Du hast jetzt ein Modell auf deinem eigenen Rechner — und damit die Gewichte in der Hand. Im nächsten Kapitel „Zensur raus? Abliteration, Heretic und die dünne Sicherheitsschicht“ siehst du, was genau passiert, wenn jemand einem solchen Modell das Nein abgewöhnt — und was es kostet. Danach: „Die KI, die sich selbst verbessert“.

So zitierst du dieses Kapitel

HerrKI: KI im eigenen Haus: lokale Modelle, VRAM, CPU gegen GPU. herrki.de/390_lokale_ki, ein privates Hobbyprojekt von Terragon.de, abgerufen am 09.09.2026.