Messprotokoll: Kolibri-1 auf dem Celsius H730

Begonnen am 6. Oktober 2026. Dieses Protokoll hält fest, was wann wie gemacht wurde. Die Rohdaten jeder einzelnen Messung liegen als JSON-Zeilen in `protokoll/`; die Zahlen hier sind daraus abgelesen. Veröffentlicht als Version 1.0 mit DOI https://doi.org/10.5281/zenodo.23222954 (Zenodo, mit Rohdaten und Werkzeug) und als Berichtsseite https://pflegedozi.org/kolibri-messbericht.html.

Autor	Benjamin Tobies, pflegedozi.org
Stand	7. Oktober 2026
Version	1.0
DOI	https://doi.org/10.5281/zenodo.23222954
Berichtsseite	https://pflegedozi.org/kolibri-messbericht.html
Lizenz	Protokoll, Rohdaten und Testtexte: CC BY 4.0. Messwerkzeug (werkzeuge/): AGPLv3.
Beilagen	Rohdaten je Anfrage als JSON-Zeilen (protokoll/), Auswertungstabellen, Messwerkzeug (werkzeuge/), Testtexte mit Quellenangaben (texte/).

Druckfassung des Protokolls, erzeugt mit werkzeuge/protokoll_pdf.py. Alle Zahlen sind aus den beiliegenden Rohdaten abgelesen; Messbedingungen und Verworfenes stehen in den Abschnitten 4, 5, 6.5, 7.1 und 10.

Inhalt

1. Ausgangslage . . . 2
2. Was installiert wurde . . . 3
3. Modelldatei . . . 4
4. Messwerkzeug . . . 5
5. Versuchsplan . . . 6
6. Ergebnisse . . . 7
   - 6.1 Tokenizer: Deutsch kostet bei Kolibri kaum mehr als Englisch . . . 7
   - 6.2 Nebenbefund: llama.cpp packt Gewichte um und braucht dafür Speicher . . . 7
   - 6.3 Nebenbefund: qwen3:30b-a3b denkt immer, und zwar auf Englisch. . . . 7
   - 6.4 Funktionstest: Kolibri läuft . . . 8
   - 6.5 Messreihe Kolibri IQ2_XS: Bedingungen . . . 8
   - 6.6 Laden, Tempo, Fäden, Ausdauer . . . 9
   - 6.7 Denkstufen . . . 9
   - 6.8 Werkzeugaufrufe . . . 10
   - 6.9 Lange Dokumente . . . 10
   - 6.10 Qualität . . . 10
   - 6.11 Wenig Arbeitsspeicher . . . 11
   - 6.12 Energie zusammengefasst . . . 11
   - 6.13 Zwischenfazit . . . 12
7. Vergleich mit qwen3:30b-a3b auf demselben Rechner . . . 13
   - 7.1 Bedingungen . . . 13
   - 7.2 Laden, Speicher, Tempo . . . 13
   - 7.3 Lange Dokumente . . . 13
   - 7.4 Umpacken (Nachprüfung zu 6.2) . . . 15
   - 7.5 Denkstufen, Werkzeuge, Qualität (über Ollama) . . . 15
   - 7.6 Tokenizer nachgeprüft . . . 17
   - 7.7 Zwischenfazit zum Vergleich . . . 17
8. Abgleich mit dem Technical Report von Aleph Alpha und mit anderen Untersuchungen . . . 18
   - 8.1 Bauweise: erklärt 6.9 und 7.3 . . . 18
   - 8.2 Tokenizer: bestätigt 6.1 und 7.6 . . . 18

8.3 Denkstufen: erklärt 6.7, Folgen für DeLoMa . . . 18
8.4 Quantisierung: keine Angaben. . . . 19
8.5 Halluzination und Faktenwissen: erklärt SIS . . . 19
8.6 Veröffentlichte Messwerte: kein Gegenstück zu unseren . . . 19
8.7 Folgen für DeLoMa 0.4 . . . 20
8.8 Andere Untersuchungen (Stand 7.10.2026, 22 Uhr) . . . 20
9. Stand am 7.10.2026, 22:15 Uhr, und nächste Schritte. . . . 22
10. Abweichungen und Verworfenes . . . 23

1. Ausgangslage

Rechner. Fujitsu Celsius H730, Intel Core i7-4900MQ (4 Kerne, 8 Fäden, AVX2), 31 GiB Arbeitsspeicher, 2 GiB Auslagerungsdatei, SATA-SSD Samsung MZ7TE512 (367 GB frei), Linux Mint 22.3, Kern 7.0.0-38. Die Grafikkarte (Quadro K2100M) wird nicht benutzt.

Modell. Kolibri-1 von Aleph Alpha, veröffentlicht am 3. Oktober 2026 unter Apache 2.0. Mixture-of-Experts mit 78,1 Mrd. Parametern, davon 3,46 Mrd. aktiv pro Token; 50 Schichten, 384 Experten je Schicht, 6 davon pro Token plus ein gemeinsamer. Trainiert auf 62,5 % Englisch, 23,9 % Deutsch und 13,6 % Code. Kontext 262 144 Tokens, laut Hersteller bis 1 048 576 geprüft.

Offizieller Weg. vLLM mit dem Paket `aleph-alpha-inference`, Mindestausstattung laut Modellkarte 2 × A100 mit 80 GB. Auf diesem Rechner nicht möglich.

Weg ohne Grafikkarte. GGUF-Dateien aus der Community für llama.cpp. llama.cpp kennt die Architektur `kolibri1` noch nicht (Anfrage ggml-org/llama.cpp#29922, am 6.10.2026 offen, kein Pull Request). Ollama, das auf llama.cpp aufbaut, kann die Dateien deshalb nicht laden. Nötig ist ein selbst gebautes llama.cpp mit einem Patch des Community-Autors Hob-forge.

2. Was installiert wurde

Alles liegt unter `~/kolibri/`, nichts davon im System:

Was	Wo	Stand
llama.cpp-Quelltext	`~/kolibri/laufzeit/llama.cpp`	Commit `836d57176dc699a726c55418e4f96b8ca628e1bf` vom 3.10.2026
Patch `kolibri1`	`~/kolibri/laufzeit/kolibri1-llama.cpp.patch`	aus `Hob-forge/Kolibri-1-GGUF`, 4 Commits, 669 Zeilen
Bauwerkzeuge	`~/kolibri/laufzeit/werkzeug-venv`	cmake 4.4.4 und ninja über pip, ohne Root-Rechte
Modelldatei	`~/kolibri/modelle/`	siehe Abschnitt 3

Der Patch wurde vor dem Bauen vollständig gelesen. Er ergänzt eine Auswahlregel für Experten (Auswahl nach Logit plus Bias, Gewichtung nach Sigmoid ohne Bias), die Modellbeschreibung `kolibri1` aus vorhandenen llama.cpp-Bausteinen, den Konverter und einen Testfall. Er enthält keine Datei-, Netz- oder Prozesszugriffe.

Bau. `cmake -G Ninja -DCMAKE_BUILD_TYPE=Release -DGGML_NATIVE=ON -DLLAMA_CURL=OFF`, nur für den Prozessor. Bauzeit auf diesem Rechner rund drei Minuten. Der mitgelieferte Architekturtest `test-llama-archs` läuft durch (134 von 134, darunter `kolibri1`).

3. Modelldatei

Gewählt wurde die kleinste Fassung, die sicher in 31 GiB passt:

Datei	`Sakura-Kolibri-1-IQ2_XS-20.94GiB.gguf`
Quelle	`webmp3/Sakura-MicroQuality-Kolibri-1-GGUF`, Stand `0fd72f818529e4697ef7942de68867c51c6e9507`
Größe	22 488 215 328 Bytes (20,94 GiB), im Mittel 2,30 Bit pro Gewicht
SHA-256	`5a55e6f918d7f045ce013df193df29083819904be0e9d80607c7f91a2840d838`

Nicht geladen, aber für 32 GB ebenfalls denkbar: Hob-forge Q2_K (26,7 GiB) und Sakura IQ3_XXS (28,8 GiB). Beide füllen den Arbeitsspeicher fast oder ganz.

Download. Gestartet am 6.10.2026 um 12:10 Uhr mit `werkzeuge/lade_modell.sh` (setzt Abbrüche fort, prüft Größe und SHA-256). Die Leitung schafft rund 1,2 MB/s; vier gleichzeitige Verbindungen brachten zusammen nicht mehr.

4. Messwerkzeug

`werkzeuge/messreihe.py <modell> <versuch> …` führt benannte Versuche aus. Jede Anfrage wird als eine Zeile mit Ergebnis und Rechnerzustand protokolliert.

- `messfuehler.py` liest einmal pro Sekunde Netzteil, Takt der arbeitenden Kerne, Temperatur, Drossel-Zähler, Energiezähler (RAPL: Prozessor und Arbeitsspeicher), freien Speicher, Auslagerung, Speicher und Seitenfehler des Modellprozesses und die Rechenlast aller anderen Prozesse („Fremdlast“).
- `laufzeit.py` spricht `llama-server` und Ollama mit gleicher Rückgabe an.
- `aufgaben.py` enthält die Aufgaben und ihre Prüfungen, `texte/` die Vergleichstexte (Herkunft in `texte/QUELLEN.md`).

Feste Einstellungen, wenn nicht anders angegeben: Kontext 4096, vier Rechenfäden, ein Platz (`--parallel 1`), Abtastwerte nach Herstellerempfehlung (Kolibri: Temperatur 1,0, top-p 0,97, top-k 128; qwen3: 0,7 / 0,8 / 20), Prompt-Zwischenspeicher aus (`--cache-ram 0`, `cache_prompt: false`), Umpacken der Gewichte aus (`--no-repack`, siehe 6.2).

Bedingungen jeder Messung: Netzteil angeschlossen (sonst bricht das Werkzeug ab), kein Download, keine anderen Programme. Messungen mit mehr als 0,25 Kernen Fremdlast gelten als gestört.

5. Versuchsplan

Versuch	Frage
`zustand`	Rechner, Programmstände, Leerlaufleistung
`tempo`	Ladezeit kalt und warm, Generierungsrate in drei Läufen
`faeden`	2, 4 oder 8 Rechenfäden
`umpacken`	Was kostet und bringt das Umpacken der Gewichte
`ausdauer`	1500 Tokens am Stück: bleibt die Rate stabil
`denken`	Drei Aufgaben auf jeder Denkstufe: Tokens, Zeit, Richtigkeit
`werkzeug`	Strukturierte Werkzeugaufrufe, auch mit den neun Werkzeugen des DeLoMa-Agenten
`kontext`	Dokumente mit 8 000, 41 000 und 75 000 Zeichen einlesen und befragen
`qualitaet`	13 deutsche Aufgaben, je zweimal
`speicherdruck`	Arbeitsspeicher auf 24, 20, 16 und 12 GB begrenzt

Vergleichsmodell ist qwen3:30b-a3b, einmal über Ollama (wie in DeLoMa und im Messbericht vom 9.9.2026) und einmal dieselbe Datei direkt in llama.cpp.

6. Ergebnisse

6.1 Tokenizer: Deutsch kostet bei Kolibri kaum mehr als Englisch

Gemessen am 6.10.2026 mit `werkzeuge/tokenizer_vergleich.py`. Dafür muss kein Modell laufen, nur sein Wortschatz. Rohdaten: `protokoll/2026-10-06_tokenizer.json`.

Text	Wörter	Kolibri-1	qwen3:30b-a3b	gemma4:e4b
Pflegebericht (erfunden)	370	593	838	828
Alltagsnachricht (erfunden)	205	254	347	338
DeLoMa-Handbuch	5 702	9 326	12 469	13 017
Startseite pflegedozi.org	981	1 369	1 936	1 992
SGB XI § 14 und § 15	1 332	2 036	3 207	3 327
Menschenrechtserklärung, deutsch	1 641	2 088	3 290	3 319
Alle deutschen Texte	**10 231**	**15 666**	**22 087**	**22 821**
Menschenrechtserklärung, englisch	1 747	2 014	2 037	2 422

qwen2.5-coder:7b benutzt denselben Tokenizer wie qwen3 und zählt gleich.

- Für dieselben deutschen Texte braucht Kolibri **29 % weniger Tokens als qwen3** und 31 % weniger als gemma4.
- Bei englischem Text liegen Kolibri und qwen3 gleichauf (2 014 zu 2 037).
- Derselbe Inhalt auf Deutsch statt Englisch kostet bei qwen3 62 % mehr Tokens, bei gemma4 37 % mehr, bei Kolibri 4 % mehr.

Was das bedeutet: Rechenzeit, Energie und Kontextverbrauch fallen pro Token an. Ein deutsches Dokument ist für Kolibri also rund 29 % „kürzer“ als für qwen3. Das ist eine Eigenschaft des Tokenizers; ob das Modell damit auch insgesamt schneller und sparsamer antwortet, zeigen erst die Messungen mit dem Modell.

6.2 Nebenbefund: llama.cpp packt Gewichte um und braucht dafür Speicher

Beim Erproben des Messwerkzeugs mit qwen3:30b-a3b (18,6 GB Datei) belegte `llama-server` 29,8 GB und drängte 1,8 GB in die Auslagerung. Ursache: llama.cpp legt Gewichte der Formate Q4_K und Q8_0 beim Laden in einer zweiten, umgepackten Fassung im Arbeitsspeicher ab (`REPACK`).

Gegenprobe mit qwen2.5-coder:7b (4,7 GB Datei), je ein Lauf:

	Speicher des Prozesses	davon nicht dateigebunden	Tok/s
Vorgabe (umpacken)	7,9 GB	3,3 GB	4,46
`--no-repack`	4,9 GB	0,3 GB	4,56

Auf diesem Prozessor bringt das Umpacken kein Tempo und kostet Speicher in der Größenordnung von zwei Dritteln der Datei. Alle Messungen laufen deshalb mit `--no-repack`; der Versuch `umpacken` prüft das für jedes Modell einzeln nach.

6.3 Nebenbefund: qwen3:30b-a3b denkt immer, und zwar auf Englisch

Die Datei, die Ollama als `qwen3:30b-a3b` führt, heißt in ihren Metadaten „Qwen3 30B A3B Thinking 2507“. Diese Fassung beginnt jede Antwort mit einem Denkblock; einen Schalter dagegen hat sie nicht.

Probe am 6.10.2026 um 15:53 Uhr über Ollama 0.33.3, Frage „Wie viele Pflegegrade gibt es in Deutschland? Antworte in einem Satz.“, je 160 Tokens:

Aufruf	Gedanken	Antwort
`think: true`	„Okay, the user is asking how many nursing care levels there are in Germany. …“	leer (nach 160 Tokens noch nicht fertig gedacht)
`think: false`	leer	„Okay, the user is asking how many nursing care levels there are in Germany. …“

Mit `think: false` denkt das Modell genauso weiter, Ollama trennt die Gedanken nur nicht mehr ab und liefert sie als Antworttext aus. Der DeLoMa-Agent ruft mit `think: false` auf (`agent_loop.py`), bekommt bei diesem Modell also englische Gedanken vor der eigentlichen Antwort. Die Gedanken sind auch bei deutscher Frage englisch.

Folgen für diese Messreihe: qwen3 wird immer mit Denken gemessen, weil es keinen anderen Modus hat. Verglichen wird deshalb nicht nur die Rate in Tokens pro Sekunde, sondern die Zeit bis zur fertigen Antwort und die Zahl der dafür erzeugten Tokens. Aufgaben, bei denen die Antwort nach einer festen Tokenzahl abgeschnitten wird, bekommen bei qwen3 einen Zuschlag von 1500 Tokens für die Gedanken.

6.4 Funktionstest: Kolibri läuft

Download am 6.10.2026 um 16:53 Uhr abgeschlossen, SHA-256 geprüft. Funktionstest um 20:55 Uhr, Kontext 4096, vier Fäden, `--no-repack`. Rohdaten: `protokoll/2026-10-06_PROBE-kolibri.jsonl`.

Anfrage	Ergebnis	Tokens	Tok/s
Laden (kalt)	89,7 s, Prozess 17,8 GB, davon nur 0,2 GB nicht dateigebunden		
„Wie viele Pflegegrade …“, ohne Denken	richtig, ein deutscher Satz	29	2,90
Tropfenrechnung, Denkstufe „low“	richtig (42), 474 Tokens Gedanken, **auf Deutsch**	481	5,45
Wetter-Werkzeug	strukturierter Aufruf `wetter_abfragen({"ort": "Heidelberg"})`	23	6,17
Folgeantwort mit Werkzeugergebnis	„Derzeit ist es in Heidelberg bewölkt mit einer Temperatur von 18 °C.“	20	5,66

- Die Gedanken beginnen mit „Ich rechne sorgfältig und prüfe am Ende jeden Schritt.“ Die Chat-Vorlage verlangt ausdrücklich, „in the user's language“ zu denken; qwen3 denkt auch bei deutscher Frage englisch.
- Die Rate steigt von Anfrage zu Anfrage (2,9 → 6,2 Tok/s): Das Modell wird nur eingeblendet, Experten kommen erst beim ersten Gebrauch von der Platte. Die Messreihe misst deshalb nach dem Aufwärmen.
- Die Fremdlast lag bei 0,22 bis 0,24 Kernen, knapp unter der Grenze von 0,25. Davon entfällt ein Teil auf den Mullvad-Dienst.

6.5 Messreihe Kolibri IQ2_XS: Bedingungen

6.10.2026, 21:01:47 bis 23:43:22 Uhr, ohne Fehler, 120 Messpunkte. Rohdaten `protokoll/2026-10-06_kolibri-iq2xs.jsonl`, Konsole `protokoll/2026-10-06_kolibri-lauf.log`, Tabellen mit `werkzeuge/auswertung.py`. Netzteil durchgehend angeschlossen, Rechner unbenutzt, Mullvad an. Leerlauf vorher: Paket 11,5 W, 0,18 Kerne Last, 56,5 °C. Drei Läufe lagen über der Fremdlastgrenze von 0,25 Kernen und sind in den Tabellen markiert (nachtdienst low 0,40 und medium 0,28, speicher_16gb_lauf2 0,35); ihre Werte weichen nicht erkennbar ab.

Datenträger: Samsung-SATA-SSD (MZ7TE512HMHP, 512 GB). Der Home-Ordner ist mit eCryptfs verschlüsselt, jede nachgeladene Seite der Modelldatei wird also auch entschlüsselt.

Wo nicht anders angegeben: Kontext 4096, vier Fäden, `--no-repack`, Abtastung nach Modellkarte (Temperatur 1,0, top_p 0,97, top_k 128, min_p 0), gemessen nach dem Aufwärmen. Energie ist immer der RAPL-Zähler des Prozessorpakets (Kerne, Grafik, Speichercontroller), ohne Bildschirm, Platte und Netzteilverluste.

6.6 Laden, Tempo, Fäden, Ausdauer

Messung	Ergebnis
Laden kalt	53,8 s, 13,0 GB tatsächlich von der Platte (der Rest lag noch im Zwischenspeicher der Verschlüsselungsschicht; nach einem Neustart eher 90 s wie im Funktionstest)
Laden warm	2,2 s
Speicher	Prozess 21,7 GB, davon nur 0,23 GB nicht dateigebunden; Zwischenspeicher für 4096 Tokens Kontext 180 MB
Antwort, 400 Tokens, 3 Läufe	**6,42–6,43 Tok/s**, Einlesen 12–14 Tok/s, erstes Antwortwort nach 4,4–5,1 s
Leistung dabei	53–54 W Paket (Leerlauf 11,5 W), 2,85–2,89 GHz, höchstens 81 °C
Energie	**8,9–9,1 J je erzeugtem Token** (Paket)
Ausdauer, 1500 Tokens am Stück	5,56 Tok/s im Mittel, 274 s, 79 °C, kein Drosseln; die Rate sinkt nur, weil der Kontext wächst

Fäden	Tok/s	J/Token
2	4,99	12,4
4	**6,77**	**9,1**
8	6,37	9,5

Vier Fäden, so viele wie echte Kerne, sind am schnellsten und am sparsamsten. Hyperthreading bringt nichts.

Umpacken: Bei IQ2_XS packt llama.cpp nur 8 MB um, mit und ohne `--no-repack` gleich schnell (6,78 und 6,85 Tok/s) und gleich groß. Der Befund aus 6.2 betrifft also Dateien mit Q4_K/Q8_0, nicht diese. `--no-repack` als Voreinstellung schadet nicht.

6.7 Denkstufen

Drei Aufgaben (Tropfrate, Dienstplanlogik, Kostformrechnung), jede Stufe einmal.

Stufe	richtig	Gedanken-Tokens	bis zur Antwort	Energie je Antwort
aus	1 von 3	0	5,5–6,1 s	0,5–0,9 kJ
low	3 von 3	72–411	17–72 s	0,9–4,6 kJ
medium	3 von 3	155–299	30–55 s	1,6–3,0 kJ
high	3 von 3	255–942	46–169 s	2,7–9,4 kJ

- Ohne Denken scheitern die beiden Rechenaufgaben, die Logikaufgabe gelingt. Schon „low“ reicht für alle drei.
- Die Gedankenlänge schwankt stark und folgt den Stufen nur grob (Kostform: low 411, medium 155 Tokens).
- **Sprache der Gedanken:** In 32 von 35 Läufen mit Gedanken (Denkstufen und Qualitätsaufgaben) denkt Kolibri auf Deutsch. Die drei Ausnahmen sind Übersetzungsaufgaben, bei denen der englische Text in der Überlegung zitiert wird.

6.8 Werkzeugaufrufe

9 von 9 Fällen richtig: jedes Mal ein strukturierter Aufruf, richtiges Werkzeug, richtige Argumente, nie nur als Text. Mit den neun echten Werkzeugen des DeLoMa-Agenten trifft es `list_directory`, `read_file` und `search_in_files` mit sinnvollen Pfaden und Suchbegriffen. Eine Wissensfrage („Hauptstadt von Österreich“) beantwortet es ohne Werkzeug. Nach dem Werkzeugergebnis antwortet es mit den 18 °C und ruft nicht erneut auf.

Dauer: 15 s für einen Aufruf mit einem Werkzeug, 45–51 s mit den neun DeLoMa-Werkzeugen. Davon gehen 44 s allein aufs Einlesen der rund 920 Tokens Werkzeugbeschreibung. Für DeLoMa heißt das: Der Zwischenspeicher für den Promptanfang muss an sein, sonst kostet jede Agentenrunde diese 44 s erneut.

6.9 Lange Dokumente

DeLoMa-Handbuch als Prompt, Frage am Anfang, in der Mitte und am Ende, dazu eine Frage nach etwas, das nicht drinsteht.

Dokument	Prompt-Tokens	Einlesen	Antwortrate	richtig
8 000 Zeichen	1 994	100 s (19,9 Tok/s)	4,8 Tok/s	ja
41 000 Zeichen	9 415	9,5 min (16,6 Tok/s)	2,3 Tok/s	ja, auch beide Folgefragen und „steht nicht drin“
75 000 Zeichen	15 754	17,9 min (14,7 Tok/s)	1,6 Tok/s	ja, auch die Folgefrage am Ende

- Alle sechs Antworten richtig, auch die Feststellung, dass eine Telefonnummer im Handbuch nicht vorkommt.
- Folgefragen zum selben Dokument dauern dank Zwischenspeicher nur 5–13 s statt Minuten.
- Der Kontextspeicher ist sehr klein, weil vier von fünf Schichten nur ein Fenster von 513 Tokens sehen: 640 MB bei 32 768 Tokens Kontext. Der Prozess wuchs bei 15 754 Tokens nur um 0,97 GB.
- Engpass ist das Einlesen. Rund 15 Tokens pro Sekunde heißen: ein Dokument von 30 Seiten braucht etwa eine Viertelstunde, bevor das erste Wort der Antwort kommt.

6.10 Qualität

13 Aufgaben aus dem Pflegealltag, je zwei Läufe, automatisch geprüft und danach von Hand nachgesehen.

Stufe	Punkte automatisch	Zeit je Antwort	Tokens gesamt
ohne Denken	103 von 114 (90 %)	36 s	3 052
low	77 von 114 (68 %)	176 s	22 143

Nachsicht von Hand, ohne Denken:

- **Unterricht, Lauf 1:** Die Prüfung fand die drei Fragen nicht, weil sie als „Frage 1“ ohne Doppelpunkt gesetzt waren. Es sind drei richtige Fragen mit je vier Optionen und Lösung, aber mit kleinen sprachlichen Fehlern („Wie lange muss die Hände desinfiziert werden“).
- **SBAR, beide Läufe:** Bei 500 Tokens abgeschnitten, bevor die Empfehlung kam. Kolibri übernimmt fast den ganzen Pflegebericht in den Hintergrund und schreibt für ein Telefonat zu lang. Der Abzug liegt am zu knappen Budget; die Gliederung bis dahin ist richtig, die Einschätzung ist mit „I“ statt „A“ beschriftet.
- **Übersetzung Englisch → Deutsch:** „Familienarzt“ für *family doctor* ist vertretbar. Lauf 1 hat zwei echte Fehler („mit Person und Ort vertraut“ statt orientiert, „welcher Tag“ statt Uhrzeit), Lauf 2 ist richtig.
- **Fachwissen:** SIS ist beide Male falsch und frei erfunden („Service-Information-System“, „Struktur-Informations-System“), statt *Strukturierte Informationssammlung*. MRSA in Lauf 1 ist inhaltlich richtig. Alle anderen Fragen sind richtig: fünf Pflegegrade, Braden-Skala, 30 s Einwirkzeit, Nüchternblutzucker.
- Nach Abzug der Prüffehler bleiben 107 von 111 bewertbaren Punkten (96 %). Echte Fehler: SIS zweimal, eine schwache Übersetzung, ein übersehener Rechtschreibfehler.

Mit Denken „low“ wird es schlechter, nicht besser:

- In 3 von 26 Läufen (Korrektur, Leichte Sprache, Fachwissen) dreht sich das Modell im Kreis und gibt gar keine Antwort. Die Gedanken wiederholen bis zu 66 Sätze („Vielleicht ‚gesichert‘? Nein.“).
- In 3 weiteren Läufen (Zusammenfassung, zweimal SBAR) ist die Überlegung fertig, aber das Budget von Antwort plus 1500 Tokens reicht nicht mehr für die Antwort.
- Aufschlussreich bei SIS: Mit Denken schreibt das Modell „Ich bin nicht sicher“, ohne Denken erfindet es eine Auflösung mit voller Überzeugung.
- Ob die Grübelschleifen am Modell liegen oder an der starken Quantisierung (2,3 Bit je Gewicht), lässt sich mit nur einer Datei nicht trennen.

6.11 Wenig Arbeitsspeicher

Das Modell lief in einer Speichergruppe mit harter Grenze (`systemd-run --user --scope -p MemoryMax=… -p MemorySwapMax=0`). Vorher wurde die Datei aus dem Zwischenspeicher geworfen. Danach folgten zwei Läufe mit je 120 Tokens auf dieselbe Frage.

Grenze	Laden	Lauf 1	Lauf 2	von Platte je Lauf	Abstürze
24 GB	60,5 s	6,51 Tok/s	6,68 Tok/s	0,1 GB	keine
20 GB	50,2 s	4,16 Tok/s	5,12 Tok/s	7,3 / 2,1 GB	keine
16 GB	46,7 s	4,96 Tok/s	5,40 Tok/s	2,9 / 0,5 GB	keine
12 GB	51,7 s	4,02 Tok/s	4,77 Tok/s	5,9 / 2,8 GB	keine

- Selbst mit 12 GB für ein Modell von 21 GB läuft Kolibri ohne Absturz, ohne Auslagerung und mit gut 4 Tokens pro Sekunde. Möglich macht das die Bauart: Je Token arbeiten nur 6 von 384 Experten. Was gerade nicht gebraucht wird, verdrängt der Kern aus dem Speicher und liest es bei Bedarf von der SSD nach.
- Das ist eine Nachbildung und kein anderer Rechner. Die Grenze gilt nur für den Modellprozess. Ein echter Rechner mit 16 GB hat nach Desktop und Browser etwa 10–12 GB für das Modell, das entspricht der Zeile 12 GB.
- Die Läufe sind kurz (120 Tokens, zweimal dieselbe Frage). Bei wechselnden Themen werden mehr Experten gebraucht, dann wird mehr nachgeladen.
- Gemessen wurde mit einer SATA-SSD. Mit einer Festplatte würde dieses Nachladen das Tempo zusammenbrechen lassen; das ist nicht gemessen.
- Energie bei 12 GB: 14,6–18,5 J/Token statt 9 J/Token.

6.12 Energie zusammengefasst

- Erzeugen: rund 9 J je Token im Prozessorpaket, also etwa 2,5 Wh für 1000 Tokens. Eine kurze Antwort ohne Denken kostet 0,5–0,9 kJ, rund 0,2 Wh.
- Denken kostet ein Vielfaches. Bei den Denkaufgaben brauchte „high“ bis zu 9,4 kJ, zehnmal so viel wie eine Antwort ohne Denken.
- Bei den Qualitätsaufgaben erzeugte „low“ siebenmal so viele Tokens wie ohne Denken (22 143 statt 3 052) bei schlechterem Ergebnis.
- Die Tokenizer-Ersparnis aus 6.1 (29 % weniger Tokens als qwen3 für denselben deutschen Text) wirkt beim Einlesen und beim Schreiben. Wie viel davon in Wattstunden ankommt, zeigt erst der Vergleich mit qwen3 auf demselben Rechner.

6.13 Zwischenfazit

Kolibri IQ2_XS ist auf einem Laptop von 2013 ohne Grafikkarte brauchbar:

- **Tempo:** 6,4 Tokens pro Sekunde, etwa so schnell, wie man liest.
- **Kurze Aufgaben ohne Denken:** fast immer richtig.
- **Werkzeuge:** zuverlässig.
- **Lange Dokumente:** richtig verstanden, aber langsam eingelesen.
- **Wenig Arbeitsspeicher:** läuft selbst bei knappem Speicher noch, solange eine SSD da ist.

Schwächen:

- erfundenes Fachwissen bei Abkürzungen, die es nicht kennt (SIS);
- Grübelschleifen mit eingeschaltetem Denken;
- das Einlesen langer Texte, mit etwa 15 Tokens pro Sekunde.

Für DeLoMa folgt daraus:

- Denken standardmäßig aus und nur für Rechenaufgaben anbieten.
- Vier Fäden, so viele wie echte Kerne.
- Kontext 16 384 Tokens; das kostet kaum Speicher.
- Zwischenspeicher für den Promptanfang an.

7. Vergleich mit qwen3:30b-a3b auf demselben Rechner

Gleicher Rechner, gleiche Laufzeit, gleiche Aufgaben und Prüfungen wie in 6.5 bis 6.9. Vergleichsmodell ist die Datei, die Ollama als `qwen3:30b-a3b` führt (Qwen3 30B A3B Thinking 2507, Q4_K_M, 4,86 Bit je Gewicht, 18,6 GB), gestartet über denselben llama-server (Commit 836d571, `--no-repack`, vier Fäden). Wie in 6.3 beschrieben, denkt dieses Modell immer; alle Zahlen hier sind Zahlen mit Denken. Wo es auf die Antwort ankommt, steht der Denkanteil dabei.

7.1 Bedingungen

- 7.10.2026, 17:08 bis 19:09 Uhr, erster Lauf nach einem Neustart. Nebenher lief nur der VPN-Tunnel; der Ollama-Dienst war gestartet, hatte aber kein Modell geladen. Energiezähler freigegeben.
- Kontext 4 096 Tokens; für die Dokumente 4 096, 16 384 und 32 768 wie bei Kolibri.
- Leerlauf vor dem Lauf: 11,65 W Paket, 0,14 Kerne, 47 °C (Kolibri-Lauf: 11,5 W).
- Versuche: zustand, tempo, kontext, umpacken. Denkstufen, Werkzeuge und Qualität laufen über Ollama (7.5).
- Rohdaten `protokoll/2026-10-07_qwen3-30b-a3b.jsonl`, Auswertung `protokoll/2026-10-07_qwen3-llamacpp-auswertung.md`, Laufprotokoll `protokoll/2026-10-07_qwen3-lauf.log`.

7.2 Laden, Speicher, Tempo

	Kolibri-1 IQ2_XS	qwen3:30b-a3b Q4_K_M
Modelldatei	22,5 GB	18,6 GB
Laden kalt	53,8 s, 13,0 GB von der Platte	36,9 s, 18,6 GB von der Platte (503 MB/s)
Laden warm	2,2 s	1,7 s
Prozess bei 4 096 Tokens Kontext	21,7 GB, davon 0,23 GB nicht dateigebunden	18,1 GB, davon 0,44 GB nicht dateigebunden
Kontextspeicher 4 096 / 32 768 Tokens	180 MB / 640 MB	384 MB / 3 072 MB
400 erzeugte Tokens, 3 Läufe	6,42–6,43 Tok/s	9,34–9,45 Tok/s
Einlesen (52 Tokens)	12–14 Tok/s	22,5–22,7 Tok/s
Leistung dabei	53–54 W, höchstens 81 °C	54,5–56 W, höchstens 72 °C
Energie je erzeugtem Token	8,9–9,1 J	6,1–6,3 J
Energie für 400 Tokens	3,2 kJ (0,89 Wh)	2,2 kJ (0,60 Wh)

Bei kurzem Kontext erzeugt qwen3 auf diesem Rechner 46 % mehr Tokens je Sekunde und braucht je Token ein Drittel weniger Energie. Die 400 Tokens der Tempo-Aufgabe waren bei qwen3 allerdings vollständig englischer Denktext („Okay, I need to explain how a heat pump works …“); eine Antwort kam innerhalb des Budgets nicht. Für Rate und Energie je Token ist das gleichgültig, für die Zeit bis zur fertigen Antwort nicht (7.3 und 7.5).

Warum das gröber quantisierte Kolibri (IQ2_XS) je Token langsamer rechnet als qwen3 (Q4_K_M), ist nicht gemessen. Eine Vermutung, keine Messung: IQ-Formate werden in llama.cpp auf dem Prozessor über Nachschlagetabellen entpackt und kosten je Gewicht mehr Rechenschritte als Q4_K.

7.3 Lange Dokumente

Dasselbe Handbuch, dieselben Fragen wie in 6.9. „Bis fertige Antwort“ ist die Dauer der ganzen Anfrage, Energie die des Prozessorpakets über diese Dauer. Der Denkanteil bei qwen3 ist aus der Zeit zwischen erstem Token und Antwortbeginn geschätzt.

Dokument	Modell	Prompt-Tokens	Einlesen	erzeugt (davon Denken)	Rate beim Erzeugen	bis fertige Antwort	Energie	richtig
8 000 Zeichen	Kolibri	1 994	100 s (19,9 Tok/s)	21 (0)	4,8 Tok/s	104 s	1,4 Wh	ja
	qwen3	2 561	121 s (21,2 Tok/s)	231 (≈187)	5,3 Tok/s	164 s	2,2 Wh	ja
41 000 Zeichen	Kolibri	9 415	9,5 min (16,6 Tok/s)	35 (0)	2,3 Tok/s	9,7 min	7,8 Wh	ja
	qwen3	12 553	17,6 min (11,9 Tok/s)	924 (≈853)	0,78 Tok/s	37,2 min	27,7 Wh	ja
75 000 Zeichen	Kolibri	15 754	17,9 min (14,7 Tok/s)	28 (0)	1,6 Tok/s	18,2 min	14,6 Wh	ja
	qwen3	22 167	45,8 min (8,1 Tok/s)	207 (≈161)	0,45 Tok/s	53,4 min	41,9 Wh	ja

Folgefragen zum selben Dokument (der Prompt liegt im Zwischenspeicher, nur die Frage ist neu):

Dokument, Folgefrage	Modell	neue Prompt-Tokens	Einlesen	erzeugt (davon Denken)	bis fertige Antwort	Energie	richtig
41 000 Zeichen, Schnellzugriff	Kolibri	20	6,2 s	14 (0)	12 s	0,15 Wh	ja
	qwen3	28	28,5 s	274 (≈229)	6,2 min	4,3 Wh	ja
41 000 Zeichen, Telefonnummer fehlt	Kolibri	17	5,4 s	22 (0)	14 s	0,18 Wh	ja
	qwen3	21	21,5 s	325 (≈282)	7,1 min	5,0 Wh	ja
75 000 Zeichen, Hausarzt	Kolibri	30	13,4 s	13 (0)	21 s	0,26 Wh	ja
	qwen3	34	59,6 s	223 (≈188)	9,2 min	6,3 Wh	ja

Befunde:

- Beide Modelle beantworten alle sechs Fragen richtig, auch die nach der Telefonnummer, die nicht im Handbuch steht.
- qwen3 braucht für denselben deutschen Text ein Drittel mehr Tokens (Tokenizer, 6.1) und liest umso langsamer ein, je länger der Kontext wird: 21, 12, 8 Tok/s gegenüber 20, 17, 15 bei Kolibri. Seine Erzeugungsrate fällt bei 22 000 Tokens Kontext auf 0,45 Tok/s, ein Zwanzigstel der Rate bei kurzem Kontext. Kolibri fällt von 6,4 auf 1,6 Tok/s.
- Folgefragen: Bei qwen3 kostet schon das Einlesen von 28 neuen Tokens 28 s, bei Kolibri 6 s. Jedes neue Token muss bei qwen3 in allen 48 Schichten den ganzen Kontext ansehen; bei Kolibri sehen vier von fünf Schichten nur 513 Tokens (6.9).
- Das Denken verdoppelt beim mittleren Dokument die Kosten: 853 Denk-Tokens und 1 089 s für eine Antwort von 71 Tokens. Aber auch ohne Denkblock bliebe qwen3 weit hinter Kolibri. Einlesezeit plus Antwort-Tokens durch Erzeugungsrate ergeben beim mittleren Dokument rund 19 min statt 9,7, beim großen rund 48 min statt 18, bei einer Folgefrage rund 1,5 min statt 12 s.
- Energie: Eine Frage zum Dokument mit 75 000 Zeichen kostet bei qwen3 das 2,9-Fache von Kolibri (41,9 statt 14,6 Wh), eine Folgefrage das 24- bis 29-Fache.
- Nebenbefund zu 6.3: Mit deutschem Dokument im Kontext dachte qwen3 auf Deutsch („Lassen Sie uns das Dokument durchsuchen …“), bei den kurzen Aufgaben auf Englisch.

Für DeLoMa: Der Vorteil von Kolibri bei langen Dokumenten liegt in der Bauweise (Fensteraufmerksamkeit, sparsamer Tokenizer), nicht in der Quantisierung. Für qwen3 gilt der RAG-Gedanke aus dem Stand (wenige passende Abschnitte statt ganzer Dokumente in den Prompt) noch dringender als für Kolibri.

7.4 Umpacken (Nachprüfung zu 6.2)

	`--no-repack`	Umpacken (Vorgabe von llama.cpp)
Laden warm	1,7 s	24,3 s
Prozess	18,1 GB, davon 0,44 GB nicht dateigebunden	30,1 GB, davon 12,6 GB nicht dateigebunden
Auslagerung	unverändert	+1,9 GB
Verfügbarer Speicher, Minimum	29,4 GB	17,7 GB
200 erzeugte Tokens	9,98 Tok/s, 6,3 J/Token	9,48 Tok/s, 6,2 J/Token
Einlesen (52 Tokens)	22,6 Tok/s	25,8 Tok/s

Bestätigt 6.2: Das Umpacken kostet bei dieser Datei 12 GB Arbeitsspeicher und 23 s Ladezeit und bringt beim Erzeugen nichts; nur das Einlesen wird um 14 % schneller. Auf einem Rechner mit 16 GB wäre qwen3 mit Umpacken nicht ladbar. `--no-repack` bleibt Voreinstellung.

7.5 Denkstufen, Werkzeuge, Qualität (über Ollama)

Lauf am 7.10.2026, 19:11 bis 21:00 Uhr, direkt im Anschluss an 7.1 bis 7.4. Ollama 0.33.3, dasselbe Modell `qwen3:30b-a3b`, `think: true`, Budgets wie bei Kolibri plus 1 500 Tokens für die Gedanken (6.3). Rohdaten `protokoll/2026-10-07_qwen3-30b-a3b-ollama.jsonl`, Auswertung `protokoll/2026-10-07_qwen3-ollama-auswertung.md`, Laufprotokoll `protokoll/2026-10-07_qwen3-ollama-lauf.log`. Ollama trennt die Gedanken von der Antwort; ihre Tokenzahl konnte das Werkzeug über Ollama nicht zählen (Abschnitt 10), die Angaben unten sind aus Zeit und Erzeugungsrate geschätzt.

Laden und Tempo über Ollama

	llama.cpp, `--no-repack` (7.2)	Ollama
Laden, Datei im Zwischenspeicher	1,7 s	20,3 s
Erneut laden nach den Tempo-Läufen	1,7 s	45,9 s, 13,9 GB von der Platte
Verfügbarer Speicher, Minimum beim Laden	29,4 GB	11,4 GB
400 erzeugte Tokens, 3 Läufe	9,34–9,45 Tok/s	9,33 / 8,50 / 8,53 Tok/s
Energie je Token	6,1–6,3 J	6,2–6,6 J

Ollama startet seinen llama.cpp-Läufer mit der Vorgabe, also mit Umpacken. Der verfügbare Speicher sank beim Laden auf 11,4 GB, und beim zweiten Laden mussten drei Viertel der Datei neu von der Platte, obwohl sie Minuten zuvor im Zwischenspeicher lag. Das passt zu 7.4: Die umgepackte Kopie (dort 12,6 GB) braucht den Platz, den die Datei im Zwischenspeicher hatte. Jeder Modellwechsel über Ollama kostet mit dieser Datei damit rund 46 s. Den Speicher des Ollama-Prozesses selbst hat das Werkzeug nicht erfasst (Abschnitt 10).

Denkaufgaben (dieselben drei wie in 6.7)

Modell, Stufe	richtig	Gedanken-Tokens	bis zur fertigen Antwort	Energie je Antwort
Kolibri, aus	1 von 3	0	5,5–6,1 s	0,5–0,9 kJ
Kolibri, low	3 von 3	72–411	17–72 s	0,9–4,6 kJ
Kolibri, high	3 von 3	255–942	46–169 s	2,7–9,4 kJ
qwen3 (denkt immer)	3 von 3	≈550–2 170	64–303 s	3,5–16,3 kJ

qwen3 löst alle drei Aufgaben, braucht dafür im Mittel 8,3 kJ je Antwort gegenüber 2,4 kJ bei Kolibri „low“ und 5,9 kJ bei „high“. Die Dienstplan-Aufgabe kostete 2 169 Tokens, davon sind fünf Zeichen Antwort („Carla“).

Werkzeugaufrufe (dieselben neun Fälle wie in 6.8)

8 von 9 richtig. Die drei Wetterabfragen und die drei Aufrufe mit den neun DeLoMa-Werkzeugen (`list_directory`, `read_file`, `search_in_files`) sind strukturiert, mit richtigem Werkzeug und richtigen Argumenten; nach dem Werkzeugergebnis nennt es die 18 °C ohne neuen Aufruf. Die Wissensfrage „Hauptstadt von Österreich“ beantwortet qwen3 nicht, sondern ruft `web_search` auf, nach eigener Überlegung bewusst („the correct action here is to use the web_search function“). Kolibri antwortete hier direkt. Für den DeLoMa-Agenten heißt das: qwen3 löst eine Websuche aus, wo Kolibri aus dem Gedächtnis antwortet.

Dauer: 29–38 s je Wetteraufruf (Kolibri 15 s), weil qwen3 vor dem Aufruf 200–350 Tokens denkt. Mit den neun DeLoMa-Werkzeugen 47 s beim ersten Aufruf, danach 20–44 s. Die Werkzeugbeschreibung ist bei qwen3 760 Tokens lang (Kolibri 920; bei englischen Bezeichnern und JSON zählt qwen3s Tokenizer günstiger), wird mit 26 Tok/s eingelesen und danach aus Ollamas Prompt-Zwischenspeicher übernommen. Energie 1,9 kJ je Aufruf (Kolibri 1,5 kJ).

Qualität (dieselben 13 Aufgaben, je zweimal, wie in 6.10)

	Punkte automatisch	Zeit je Antwort	Tokens gesamt	Energie gesamt
Kolibri ohne Denken	103 von 114 (90 %)	36 s	3 052	12,2 Wh
Kolibri low	77 von 114 (68 %)	176 s	22 143	59,7 Wh
qwen3 (denkt immer)	42 von 114 (37 %)	210 s	37 973	71,1 Wh

Die 37 % sind vor allem ein Budgetbefund. In 13 von 26 Läufen (Zusammenfassung 2×, Korrektur 2×, Leichte Sprache 2×, Übersetzung EN→DE 2×, SBAR 2×, Übersetzung DE→EN, Unterricht, Programm) war das Budget von Antwort plus 1 500 Tokens aufgebraucht, bevor die Antwort begann; sie bekamen 0 Punkte (die zwei Punkte bei Korrektur sind Prüfkriterien, die auch ein leerer Text erfüllt). Anders als bei Kolibri „low“ sind das keine Grübelschleifen: In den 13 abgebrochenen Gedankenblöcken (6 400 bis 8 700 Zeichen) wiederholen sich höchstens 11 von 144 Sätzen. Das Modell entwirft die komplette Antwort im Denkblock, prüft sie Punkt für Punkt und kommt nicht mehr zum Aufschreiben. Mit einem Denkbudget von rund 3 000 Tokens, bei 7 Tok/s sieben Minuten je Antwort, wären diese Antworten vermutlich gekommen; gemessen ist das nicht.

Wo eine Antwort kam (13 Läufe), erreicht qwen3 automatisch 40 von 46 Punkten (87 %). Nachsicht von Hand:

- **Fachwissen, beide Läufe:** „Die Braden-Skala schätzt das Risiko für Druckgeschwüre ein“ ist richtig, die Prüfung suchte „Dekubitus“ (Prüffehler, 2 Punkte). SIS ist wie bei Kolibri zweimal erfunden („Sicherheits- und Informations-System“, „Sicherheits-Informationssystem“). Einwirkzeit 20 s statt der erwarteten 30 s (RKI); die WHO nennt 20–30 s, die Antwort ist knapp, nicht falsch.
- Extraktion (JSON), Rechnen, beide Dokumentfragen, Übersetzung DE→EN (Lauf 1), Unterricht (Lauf 2), Programm (Lauf 1): alle Punkte.
- Nach Abzug des Prüffehlers 42 von 46 (91 %). Echte Fehler: SIS zweimal, Einwirkzeit zweimal knapp.

Wo qwen3 antwortet, liegt die Qualität auf dem Niveau von Kolibri ohne Denken (96 % nach Nachsicht), mit demselben erfundenen Fachbegriff. Der Unterschied ist der Preis: 9,8 kJ und 210 s je Antwort gegenüber 1,7 kJ und 36 s, das Sechsfache an Energie und Zeit, bei halb so vielen fertigen Antworten.

Sprache der Gedanken: Alle 38 Gedankenblöcke dieses Laufs beginnen auf Englisch („Okay, let me tackle this …“), auch bei deutschen Aufgaben mit deutschen Antworten. Nur mit einem langen deutschen Dokument im Kontext (7.3) dachte das Modell auf Deutsch.

7.6 Tokenizer nachgeprüft

6.1 zählte Kolibris Tokens mit Aleph Alphas `tokenizer.json`, weil die GGUF-Datei noch fehlte. Nachzählung am 7.10.2026 mit `llama-tokenize` aus der GGUF: 15 664 statt 15 666 Tokens für alle deutschen Texte (10 231 Wörter), beim Handbuch 9 324 statt 9 326. Die Zählung aus 6.1 steht. qwen3 braucht für dieselben Texte 22 087 Tokens, 41 % mehr als Kolibri; umgekehrt gerechnet sind das die 29 % weniger aus 6.1. Protokoll `protokoll/2026-10-07_tokenizer.json`.

7.7 Zwischenfazit zum Vergleich

Auf dem Celsius H730 ohne Grafikkarte:

	Kolibri-1 IQ2_XS	**qwen3:30b-a3b Q4_K_M (Thinking)**
Erzeugen, kurzer Kontext	6,4 Tok/s, 9 J/Token	9,4 Tok/s, 6,2 J/Token
13 Alltagsaufgaben, je fertige Antwort	36 s, 1,7 kJ; 90 % (96 % nach Nachsicht)	210 s, 9,8 kJ; 37 %, davon 91 % wo geantwortet
Rechen- und Logikaufgaben	mit „low“ 3 von 3, 2,4 kJ je Antwort	3 von 3, 8,3 kJ je Antwort
Werkzeuge	9 von 9	8 von 9
Dokument 75 000 Zeichen, eine Frage	18 min, 14,6 Wh	53 min, 41,9 Wh
Folgefrage dazu	21 s, 0,26 Wh	9 min, 6,3 Wh
Speicher	Prozess 21,7 GB; läuft mit SSD auch bei 12 GB (6.11)	18,1 GB mit `--no-repack`, 30,1 GB mit Umpacken (7.4); Ollama packt um
Denken abschaltbar	ja	nein

- Je Token ist qwen3 das schnellere und sparsamere Modell. Aber es denkt immer, und das Denken frisst den Vorsprung: je fertiger Antwort das Sechsfache an Energie und Zeit, und mit dem Kolibri-Budget kommt nur die Hälfte der Antworten überhaupt an.
- Bei langen Dokumenten ist Kolibri durch die Bauweise klar im Vorteil (Fensteraufmerksamkeit in vier von fünf Schichten, 29 % weniger Tokens): dreimal so schnell bei einer Frage, rund 25-mal bei Folgefragen.
- Wo beide antworten, ist die Qualität ähnlich, mit derselben Schwäche (SIS erfunden).
- Für DeLoMa: Kolibri als Vorgabe für Dokumentarbeit und Alltagsaufgaben, Denken aus (6.13). Für qwen3 Thinking ein Denkbudget von mindestens 3 000 Tokens, sonst kommen Antworten nicht an. Umpacken aus. Websuche nur bewusst freigeben. RAG für beide Modelle, für qwen3 dringender.

8. Abgleich mit dem Technical Report von Aleph Alpha und mit anderen Untersuchungen

Quelle: „Kolibri: A Sovereign European Model on the Pareto Frontier“, Aleph Alpha, 3.10.2026, 189 Seiten, https://aleph-alpha.com/downloads/tech-report.pdf. Geladen am 7.10.2026 nach `report/kolibri-1-tech-report.pdf`, SHA-256 `01520e07506e67d53aebfb16b5298384940870d652409ea3943fe990f7a68b0e`. Gezielt gelesen wurden die Seiten 7–15 (Bauweise, Tokenizer), 49–52 (Denkstufen), 78 und 183 (Budgetsignal), 98–103 (Ergebnisse, Halluzination), 109–110 (Zusammenfassung), 130–133 (Durchsatz, Tokenizer-Anhang) und 167 (Denken bei langem Kontext). Seitenangaben sind PDF-Seiten.

8.1 Bauweise: erklärt 6.9 und 7.3

- 50 Blöcke; 40 mit Fensteraufmerksamkeit über die 512 vorangehenden Tokens plus das aktuelle (unsere „513“ aus 6.9 stimmt), jeder fünfte Block mit voller Aufmerksamkeit ohne Positionskodierung (S. 7–8, 12). Grouped-Query Attention mit 48 Abfrage- und 4 KV-Köpfen, Kopfdimension 128.
- Kontextspeicher: Nur die 10 vollen Blöcke wachsen mit dem Kontext. 10 × 2 × 4 × 128 × 2 Byte = 20 KiB je Token, bei 32 768 Tokens 640 MiB. Genau das haben wir gemessen (6.9: 640 MB). qwen3 hat 48 volle Schichten, daher 3 072 MiB (7.3).
- Der Report sagt selbst (S. 15): Bei langem Kontext kommt fast der ganze Speicherverkehr aus den 10 vollen Schichten, die 40 Fensterschichten lesen ein festes Fenster und „tragen vernachlässigbar bei“. Das ist der Mechanismus hinter 7.3 (Folgefrage 21 s statt 9 min).
- Mixture of Experts: 384 Experten je Schicht, 6 geroutete plus 1 gemeinsamer je Token; 78,1 Mrd. Gewichte gesamt, 3,46 Mrd. aktiv (4,4 %) (Tabelle 2, S. 8). qwen3:30b-a3b hat rund 3,3 Mrd. aktive. Der Tempounterschied je Token (6,4 zu 9,4 Tok/s) kommt also nicht aus der Modellgröße, sondern aus Quantisierung (IQ2_XS gegen Q4_K_M) und Umsetzung in llama.cpp.
- Die Fenstergröße 512 wurde gegen 128 bis 4 096 ausgemessen; 512 verliert bei 256k Kontext am wenigsten und kostet 0,2 Punkte auf den Vortrainings-Benchmarks (S. 13).

8.2 Tokenizer: bestätigt 6.1 und 7.6

- UniBPE, 128 000 Tokens, jede Ziffer einzeln, Byte-Rückfall (S. 10). Auf deutschem Webtext 4,90 Byte je Token gegenüber 4,35 bei GPT-5, dem besten der neun fremden Tokenizer; „11,2 % weniger Tokens als GPT-5“ (S. 10, 109).
- Fertility (Tokens je Wort, 1 000 deutsche Wörter aus Häufigkeitslisten, Tabelle 30, S. 132): Kolibri 1,87, Qwen3-Next 3,20, Qwen3.5–3.8 2,60, Gemini/Gemma 4 2,59, GPT-5 2,56.
- Unsere Zählung an laufendem Text (7.6): Kolibri 1,53 Tokens je Wort, qwen3 2,16, gemma4 2,23. Die Richtung stimmt, der Abstand ist bei uns kleiner (qwen3 +41 % statt +71 %), weil Einzelwortlisten seltene lange Wörter überbetonen und laufender Text Satzzeichen und Zahlen enthält, die alle Tokenizer ähnlich zerlegen. Beide Messungen gelten, sie messen Verschiedenes.
- 65 % der wortinternen Tokengrenzen liegen im Deutschen auf Morphemgrenzen, bei den anderen Tokenizern höchstens 49 % (S. 133).

8.3 Denkstufen: erklärt 6.7, Folgen für DeLoMa

- Vier Stufen: none, low, medium, high (Tabelle 19, S. 50). Die Chat-Vorlage schreibt dafür einen Satz in den Systemprompt unter „# Reasoning effort“, für low: „Reasoning effort is set to low. Think briefly through only the essential steps in the user's language, then proceed directly to the answer.“ Der Wortlaut ist über die API nicht änderbar (S. 51).

- „in the user's language“: Deshalb dachte Kolibri in 32 von 35 Läufen auf Deutsch (6.7). Im Training wurde das mit einem deutschen Satzanfang vor jeder Lehrer-Denkspur und einer Sprachkonsistenz-Belohnung im RL verankert (S. 49).
- **Voreinstellung ist high.** Die Anfragewerte high, xhigh, max und „default“ landen alle bei high (Tabelle 19). Wer nichts setzt, bekommt die längste Denkstufe. Sind enable_thinking und reasoning_effort zugleich gesetzt, gewinnt reasoning_effort (S. 51).
- Ein Budget zur Laufzeit gibt es nicht. Das „completion_budget“ ist ein Belohnungssignal im RL-Training: Die Belohnung sinkt, sobald die Ausgabe länger wird als 0,2 / 0,4 / 0,8 des Budgets für low / medium / high (Tabelle 24, S. 78; Tabelle 59, S. 183). Im Retrieval-Training fällt sie linear mit den Denk-Tokens und ist bei 16 384 null. Die Stufen sind das einzige Stellrad; ein hartes Abschneiden wie unser max_tokens kennt das Modell aus dem Training nicht.
- Zu Grübelschleifen (6.10) sagt der Report nichts; die Treffer zu „repetition“ betreffen die Datenbereinigung. Ob die Schleifen am Modell oder an IQ2_XS liegen, bleibt offen.
- Denken bei langem Kontext (Anhang G, S. 167): Mit Denken aus ist die Nadel-im-Heuhaufen-Trefferquote höher, mit Denken an die Fragebeantwortung bei den längsten Kontexten; Aleph Alpha verweist auf denselben Befund im Qwen3-Report. Das stützt unsere Vorgabe „Denken aus“ für Dokumentarbeit (6.13).
- Figur 20 (S. 52): Die mittlere Ausgabelänge auf Benchmarks steigt von none bis high auf bis zu 12 000 Tokens. Unsere Budgets (Antwort plus 1 500) passen zu none und low, nicht zu high.

8.4 Quantisierung: keine Angaben

Veröffentlicht sind FP8-Gewichte (78,1 Mrd. × 1 Byte, rund 78 GB); alle Durchsatzmessungen des Reports laufen in FP8 mit FP8-Kontextspeicher auf acht B200 (S. 130). Zu GGUF, 4 Bit oder 2 Bit steht nichts im Report. Unsere IQ2_XS-Datei (22,5 GB, rund 2,3 Bit je Gewicht) ist eine Community-Quantisierung weit unterhalb von allem, was Aleph Alpha bewertet hat. Die Qualitätsabschläge aus 6.10 lassen sich mit dem Report weder dem Modell noch der Quantisierung zuordnen.

8.5 Halluzination und Faktenwissen: erklärt SIS

- Kolibris schwächste Zeilen im Vergleich mit zwölf Modellen (S. 99): RGB Closed-Book, also Wissen ohne Dokument, auf dem letzten Platz (51,0); AA-Omniscience-Genauigkeit 14,8 %; RGB Fact-Check 34,0. Stark dagegen beim Antworten aus Dokumenten und beim Zurückhalten: AA-Omniscience Non-Hallucination 44 % (Kolibri Origin 15 %), RGB Negative (Abstehen) 85,6 %, M/A Grounding Score 23,4 % wo mehrere andere Modelle 0 haben (S. 101–103). Trainiert mit dem Merlin–Arthur-Protokoll: belohnt wird die richtige Antwort aus dem Kontext und das Abstehen, wenn der Kontext die Antwort nicht enthält (S. 50).
- Das ist genau unser Bild: SIS aus dem Gedächtnis zweimal erfunden (6.10), aber „Telefonnummer steht nicht drin“ (6.9) und „dokument_fehlt“ (6.10) jedes Mal richtig. Der Report bestätigt die Richtung aus dem Stand: Kolibri gehört mit Quellen gefüttert (RAG), nicht als Nachschlagewerk befragt.

8.6 Veröffentlichte Messwerte: kein Gegenstück zu unseren

- Der Report misst Durchsatz als Summe über viele gleichzeitige Anfragen auf acht B200 in FP8 (S. 130–131), in der Größenordnung von tausenden Tokens je Sekunde und GPU (Figur 50). Einzelnutzer-, CPU- oder Energiewerte gibt es nicht. Unsere Zahlen (6,4 Tok/s, 9 J je Token auf einem Laptop von 2013) haben im Report kein Gegenstück; sie ergänzen ihn.
- Benchmarks (Tabellen 28 und 29, S. 100–101): Kolibri Overall Englisch 75,5, Deutsch 70,8; die 30B-A3B-Klasse (Nemotron 3 Nano, GLM-4.7 Flash) liegt bei 62–66 und 50–59. Unser qwen3:30b-a3b (Thinking 2507) steht nicht in der Tabelle.

- Sampling: Die Modellkarte empfiehlt temperature 1,0, top_p 0,97, top_k 128; unser Werkzeug hat genau damit gemessen (`messreihe.py`, KOLIBRI_ABTASTUNG), qwen3 mit den Qwen-Empfehlungen 0,7 / 0,8 / 20. Im RL-Training wurde mit temperature 1 und top-p 1 gesampelt (Tabelle 59, S. 183).

8.7 Folgen für DeLoMa 0.4

1. Denkstufe immer ausdrücklich setzen (none oder low); nie die Vorgabe der Chat-Vorlage (high) durchreichen.
2. Denken aus für Dokumentarbeit, low nur für Rechnen und Logik (bestätigt 6.13).
3. RAG mit Abschnittsauswahl: spielt Kolibris Stärke aus (Antwort aus Quellen, Abstehen) und umgeht die Schwäche (geschlossenes Wissen).
4. Kontext großzügig bemessen: 20 KiB je Token, 32 768 Tokens sind 640 MiB.
5. Sampling-Werte der Modellkarte als Vorgabe für `gguf:`-Modelle übernehmen.

8.8 Andere Untersuchungen (Stand 7.10.2026, 22 Uhr)

Gesucht auf Hugging Face, GitHub und im Netz. Vier Tage nach der Veröffentlichung gibt es 25 abgeleitete Modelle (9 GGUF, 7 MLX, der Rest NVFP4, FP8 und beschnittene Fassungen), aber keine Veröffentlichung, die ein Kolibri-Modell auf alter oder kleiner Hardware systematisch misst, und keine mit Energiewerten. Unterstützung: Die llama.cpp-Anfrage ggml-org/llama.cpp#29922 ist weiter offen, ohne Pull Request; in der Diskussion „llama.cpp support?“ auf der Modellkarte (huggingface.co/Aleph-Alpha/Kolibri-1/discussions/3) hat sich Aleph Alpha nicht geäußert. Ollama und LM Studio können die Dateien weiter nicht laden. Alle GGUF-Anbieter setzen wie wir auf den Hob-forge-Patch auf Commit 836d571.

Unsere Quellen, Stand heute: GGUF `Sakura-Kolibri-1-IQ2_XS` von webmp3 (huggingface.co/webmp3/Sakura-MicroQuality-Kolibri-1-GGUF). Die Modellkarte nennt drei Dateien: IQ2_XS 20,94 GiB mit 2,30 Bit je Gewicht, IQ3_XXS 28,75 GiB mit 3,16, IQ4_XS 38,07 GiB mit 4,19. KL-Divergenz gegen Q8_0 um 59 / 53 / 40 % niedriger als die gleich großen Standardquantisierungen, 90–95 % gleiche Top-Token; ausdrücklich nur KL und Perplexität gemessen, keine Aufgaben; rund 5 % der Experten in frühen Schichten wurden bei der Kalibrierung nie ausgewählt. Patch von Hob-forge (huggingface.co/Hob-forge/Kolibri-1-GGUF).

Vergleichbare Messwerte anderer (Erzeugungsraten, jeweils ein Nutzer):

Wer	Rechner	Datei	Erzeugen	Einlesen
Hob-forge, Modellkarte	Ryzen 7 7800X3D, 8 Fäden, 128 GB DDR5, keine GPU	Q4_K_M, 47,5 GB	12,3 ± 1,4 Tok/s	83 ± 13 Tok/s
Hob-forge	dazu RTX 5070 (12 GB), Experten im RAM	Q4_K_M	18,0 Tok/s	723 Tok/s
cmgenerate (Rust-Engine, A. Holzinger, 4.10.)	32-Kern-CPU, 256 GB	Q4_K / FP8	19,6 Tok/s	
cmgenerate	16-Kern-Mobilprozessor		6,8 Tok/s	
Nutzer ulymp, HF-Diskussion	RTX 3090, llama.cpp-Fork moe-cache	4-Bit-GGUF	90 Tok/s	
Berichte aus zweiter Hand	Apple Silicon ab 48 GB	3 bis 4 Bit	50–116 Tok/s	
wir (6.6)	**i7-4900MQ von 2013, 4 Fäden, 31 GiB DDR3, keine GPU**	**IQ2_XS, 22,5 GB**	**6,4 Tok/s, 9 J je Token**	**12–15 Tok/s**

Einordnung: Der nächstliegende Vergleich ist Hob-forges 7800X3D ohne GPU. Zehn Jahre jüngerer Prozessor, DDR5 statt DDR3, doppelt so viele Fäden: doppelte Erzeugungsrate, sechsfache Einleserate. Dass das Einlesen bei uns relativ am schwächsten ist, passt dazu, dass Einlesen rechen- und Erzeugen speichergebunden ist; der Rechenabstand zwischen Haswell-AVX2 und Zen 4 ist größer als der Bandbreitenabstand. Niemand sonst misst Energie, niemand hat weniger als 32 GB Arbeitsspeicher getestet, niemand prüft die 2-Bit-Dateien an Aufgaben.

Quantisierung, Teilantwort auf die offene Frage aus 6.10: Hob-forge misst die KL-Divergenz zur Referenz je Stufe: Q4_K_M 0,013, Q3_K_M 0,034, Q2_K 0,129. Zwei Bit kosten demnach das Zehnfache an Abweichung gegenüber vier Bit; die Sakura-Datei halbiert das für ihre Größe. Dass IQ2_XS an den Grübelschleifen und am erfundenen SIS beteiligt ist, wird damit wahrscheinlicher, ist aber nicht belegt. Prüfbar ist es mit Sakura IQ3_XXS (28,75 GiB), die gerade noch in 31 GiB passt (6.11 zeigt, dass Kolibri Speicherdruck mit SSD übersteht): dieselben 13 Qualitätsaufgaben ohne Denken und mit „low“ sowie die drei Denkaufgaben, dann der Vergleich zu 6.7 und 6.10.

Langkontext: Hob-forge findet mit Q4_K_M drei versteckte Fakten bei 16 463 und 65 915 Tokens auf der CPU (3 von 3), mit GPU bis 253 646 Tokens; Denken aus, Temperatur 0, ohne Zeitangaben. Unsere 6.9 ist die einzige Messung mit Einlesezeiten für lange deutsche Dokumente.

Beschnittene Fassungen für kleine Rechner:

- webmp3 „365E“: 19 wenig genutzte Experten je Schicht entfernt (0,07 % der Zuweisungen), IQ2_XS 19,99 GiB, nur 1 GiB kleiner als unsere Datei. Lohnt für uns nicht.
- MediSync-AI REAP-31B-A3B-W4A16: 144 von 384 Experten behalten, 17,2 GB, nur vLLM auf NVIDIA. Medizinische Benchmarks: MedQA Englisch 78,0 vor und 66,8 nach dem Beschneiden, ACI-Bench ROUGE-L 45,0 zu 44,8. Für uns nicht lauffähig, aber ein Maß dafür, was Expertenbeschneiden an Fachwissen kostet.

Andere Laufzeiten: DwarfStar ds4 (github.com/aparusel/ds4-kolibri-1, Fork von antirez' Engine) läuft auf CPU und Apple Metal mit Kolibri-Unterstützung, eigener 2-Bit-Mischung (IQ2_XXS und Q2_K, rund 23 GiB) und KV-Ringen für die Fensterschichten: 12 GiB Kontextspeicher bei 262 144 Tokens. cmgenerate (Rust) läuft auf CPU und NVIDIA. Beide ohne CPU-Messreihen. Für DeLoMa bleibt llama.cpp mit Patch der Weg, weil Ollama darauf aufbaut und `llama_laufzeit.py` dafür geschrieben ist.

Schluss: Soweit auffindbar sind wir vier Tage nach der Veröffentlichung die Einzigen, die ein Kolibri-Modell auf einem Rechner unter 32 GB ohne GPU systematisch messen, Energie je Token angeben, die 2-Bit-Datei an Aufgaben statt nur an KL und Perplexität prüfen und Einlesezeiten für lange deutsche Dokumente messen. Das ist der Beitrag der Berichtsseite.

9. Stand am 7.10.2026, 22:15 Uhr, und nächste Schritte

Fertig: Laufzeit, Messwerkzeug, Tokenizer-Vergleich (6.1, nachgeprüft in 7.6), Download und Prüfung der Modelldatei, Funktionstest (6.4), komplette Messreihe Kolibri IQ2_XS mit Auswertung und Nachsicht von Hand (6.5–6.13), Vergleich mit qwen3 in llama.cpp und über Ollama mit Zwischenfazit (7.1–7.7), Abgleich mit dem Technical Report und Suche nach anderen Untersuchungen (8).

Nächste Schritte, einer nach dem anderen:

1. Berichtsseite: am 7.10.2026 geschrieben als `DeLoMa v0.3/website-upload/kolibri-messbericht.html` (Stil von messbericht.html, drei SVG-Diagramme ohne Skripte, englische Kurzfassung am Ende), verlinkt aus messbericht.html, deloma.html und pflegedoku-pro.html. Am 7.10.2026 um 22:36 Uhr hochgeladen (vier Dateien byte-gleich geprüft). Zeitstempel im Internet Archive, 7.10.2026 20:39 UTC: https://web.archive.org/web/20261007203906/https://pflegedozi.org/kolibri-messbericht.html und https://web.archive.org/web/20261007203926/https://pflegedozi.org/deloma.html.
2. Veröffentlichung außerhalb von pflegedozi.org (Entscheidung vom 7.10.2026): ohne neue Konten (Entscheidung vom 7.10.2026, abends): Mastodon deutsch und englisch, Threads, Instagram-Kachel, E-Mail mit PDF an Aleph Alpha, E-Mail an heise online mit beiden Themen (Kolibri-Messung und PflegeDoku Pro); Hugging-Face-Diskussionen, llama.cpp #29922 und r/LocalLLaMA nur, falls dafür doch ein Konto angelegt wird (Texte liegen in `veroeffentlichung/kurztexte-2026-10-07.md`). Zenodo-DOI 10.5281/zenodo.23222954 (am 7.10.2026 reserviert; PDF-Fassung mit `werkzeuge/protokoll_pdf.py --doi` erzeugt), Fediverse. Eine kanonische Fassung auf pflegedozi.org, sonst nur Link. Bei Nachfragen zur Sinnhaftigkeit der 2-Bit-Datei: Bezug zum möglichen Einsatz mit PflegeDoku Pro herstellen, also Pflegedokumentation auf vorhandenen alten Rechnern in Einrichtungen, wegen der Schweigepflicht lokal und ohne Cloud; die kleinste lauffähige Datei ist dort die praktische Frage, nicht eine akademische.
3. DeLoMa 0.4 weiterbauen (Liste in `DeLoMa v0.4/STAND-2026-10-06.md`), mit den Standardwerten aus 6.13 und den Folgen aus 8.7. Ziel: Mitte Oktober 2026, als zusätzlicher Download neben 0.3 mit der Kurzbezeichnung „für Kolibri“; so am 7.10.2026 auf der Berichtsseite und auf deloma.html angekündigt. Testsuite am 7.10. mit der Python-Umgebung des Installers: 194 von 194 bestanden.
4. Vor der nächsten Messreihe das Messwerkzeug nachbessern (Befunde in Abschnitt 10): Anzeige „Datei im Speicher“ bei Dateien ohne Schreibrecht, Kaltstart und Prozessspeicher für `@ollama`-Modelle, Gedanken-Tokens ohne llama-server schätzen.

Mögliche Nachmessungen: Sakura IQ3_XXS (28,75 GiB) auf demselben Rechner mit den Qualitäts- und Denkaufgaben, um den Anteil der Quantisierung an Grübelschleifen und SIS zu bestimmen (8.8); die 13 abgebrochenen Qualitätsaufgaben mit qwen3 und Denkbudget 3 000 Tokens; SBAR mit größerem Budget bei Kolibri; Denken „low“ mit größerem Budget, um Grübelschleifen von abgeschnittenen Antworten sauber zu trennen; qwen3 bei begrenztem Arbeitsspeicher (Gegenstück zu 6.11). Energiezähler nach jedem Neustart freigeben (Befehl in Abschnitt 10).

10. Abweichungen und Verworfenes

- **7.10., 19:11–21:00 Uhr, qwen3 über Ollama, Gedanken-Tokens:** Das Werkzeug zählt Gedanken-Tokens über den eigenen llama-server; über Ollama gibt es den nicht, die Spalte zeigt 0. In 7.5 sind die Gedanken aus der Zeit zwischen erstem Token und Antwortbeginn mal Erzeugungsrate geschätzt; bei abgebrochenen Läufen sind alle erzeugten Tokens Gedanken.
- **Budgetregel aus 6.3 (Antwort plus 1 500 Tokens) reicht für qwen3 nicht:** 13 von 26 Qualitätsantworten abgebrochen. Kein Messfehler, aber eine Grenze des Versuchsplans; Nachmessung mit größerem Budget steht in Abschnitt 9.
- **7.10., 17:08–19:09 Uhr, qwen3 in llama.cpp, Spalte „Datei im Speicher“:** zeigt in allen Zeilen 17 697 MB, also die ganze Datei, auch unmittelbar nach dem Leeren des Zwischenspeichers. Die Modelldatei gehört dem Benutzer `ollama`; für Dateien ohne Schreibrecht meldet der Kern über `mincore` seit Version 5.1 absichtlich alle Seiten als geladen (Schutz vor Seitenkanalangriffen). Der Wert ist für diesen Lauf unbrauchbar. Dass der Kaltstart echt war, zeigt das Plattenlesen: 18,6 GB in 36,9 s. Bei Kolibri lag die Datei im eigenen Heimatordner, dort stimmte die Anzeige („Kalt: 0 MB“).
- **7.10., 17:12 Uhr, kontext_klein:** Fremdlast 0,48 Kerne während der Anfrage, Ursache nicht bekannt, kein eigener Eingriff zu der Zeit. Einlese- und Erzeugungsrate liegen im Rahmen der Nachbarwerte; der Wert bleibt im Bericht.
- **7.10., 19:11 Uhr, qwen3 über Ollama, „Laden kalt“:** war kein Kaltstart. Das Werkzeug wirft für `@ollama`-Modelle die Datei nicht aus dem Zwischenspeicher, und sie lag aus dem llama.cpp-Lauf noch dort (1,1 GB von der Platte statt 18,6). Die 20,3 s sind Ollamas Startaufwand bei warmem Zwischenspeicher. Außerdem erfasst das Werkzeug Speicher und Fremdlast des Ollama-Prozesses nicht (`None`), und die Einleserate der Läufe 2 und 3 (rund 500 Tok/s) ist ein Treffer im Prompt-Zwischenspeicher von Ollama, kein Einlesen.
- **6.10., 12:28–12:50 Uhr, Probe 2 mit qwen3 in llama.cpp:** ebenfalls nur Funktionsprobe, lief neben dem Download. Ergebnis: Werkzeugaufrufe kommen strukturiert an, ohne Umpacken lädt die Datei warm in 3,3 s statt 27,8 s und belegt 18,1 GB statt 29,8 GB. Datei `protokoll/2026-10-06_PROBE2-qwen3-30b-a3b.jsonl`.
- **6.10., 12:16–12:20 Uhr, Probe 1 mit qwen3 in llama.cpp:** nur Funktionsprobe, nicht verwertbar. Der Download lief nebenher (Fremdlast 0,6 bis 0,7 Kerne), das Denken des Modells war wegen eines falschen Schalters nicht abgeschaltet, und das Umpacken war noch aktiv. Datei `protokoll/2026-10-06_PROBE-qwen3.jsonl` bleibt als Beleg liegen.
- **Energiezähler:** Die RAPL-Zähler sind seit Kern 5.10 nur für Root lesbar. Sie müssen nach jedem Neustart einmal freigegeben werden: `sudo chmod a+r /sys/class/powercap/intel-rapl:0/energy_uj /sys/class/powercap/intel-rapl:0:*/energy_uj`. Ohne Freigabe misst das Werkzeug alles außer Energie.