Technik & Architektur

Anleitung: Ollama mit Open WebUI, ein KI-Chat für das eigene Team



Mit KI erzeugtes Bild. Mehr zur KI-Transparenz

Was am Ende läuft

Ein Rechner im Büro beantwortet Fragen wie ChatGPT, nur dass kein Text das Haus verlässt. **Ollama** lädt das Sprachmodell und rechnet, **Open WebUI** ist die Weboberfläche davor, mit Anmeldung, Gesprächsverlauf und Benutzerverwaltung. Die Kolleginnen und Kollegen öffnen einfach eine Adresse im Browser.

Ollama und die Modelle haben wir am 25. Oktober 2026 auf einem PC mit AMD Ryzen 9 5900X, 64 GB Arbeitsspeicher und einer NVIDIA RTX 3090 (24 GB) unter Windows 11 Pro gemessen, mit Ollama 0.40.11. Die Schritte für Linux, für Open WebUI und für die Freigabe im Netzwerk folgen der offiziellen Dokumentation (Stand 24. Oktober 2026). Ohne Grafikkarte geht es auch, dann aber deutlich langsamer; unsere Messung auf einem Server ohne Grafikkarte zeigt, womit man dann rechnen muss.

Was ihr braucht

- Einen Rechner, der im Netzwerk erreichbar ist und laufen darf, am besten mit einer NVIDIA-Grafikkarte ab 12 GB Grafikspeicher. Wie viel Speicher welches Modell braucht, steht in unserer Tabelle.
- Windows 10 (22H2) oder neuer, oder ein aktuelles Linux. Für NVIDIA unter Windows mindestens Treiber 551.61.
- Rund 4 GB Platz für Ollama selbst und je Modell zwischen 5 und 50 GB.
- Für Open WebUI: Docker (unter Windows Docker Desktop) oder Python 3.11 bzw. 3.12.
- Bei AMD-Karten: unter Windows läuft Ollama über ROCm 7 oder über Vulkan, das ab Werk eingeschaltet ist. Einige Karten der RX-6000-Reihe bieten mit aktuellen Treibern kein ROCm 7, für sie ist Vulkan der Weg. Mehr dazu unter ROCm und Vulkan.

Schritt 1: Ollama installieren

Unter Windows

1. Den Installer `OllamaSetup.exe` von ollama.com herunterladen und starten. Administratorrechte braucht es nicht, Ollama landet im eigenen Benutzerordner.
2. Nach der Installation läuft Ollama im Hintergrund, erkennbar am Symbol im Infobereich der Taskleiste.
3. Eine Eingabeaufforderung oder PowerShell öffnen und prüfen:

```
ollama -v
```

Die Modelle liegen unter `%HOMEPATH%\.ollama`. Wer sie auf eine andere Platte legen will, setzt in den Umgebungsvariablen des eigenen Kontos `OLLAMA_MODELS` auf den gewünschten Ordner und startet Ollama danach neu.

Unter Linux

```
curl -fsSL https://ollama.com/install.sh | sh
ollama -v
```

Das Skript richtet Ollama als Dienst `ollama` ein. Die Modelle liegen unter `/usr/share/ollama/.ollama/models`, das Protokoll zeigt `journalctl -e -u ollama`. Für NVIDIA muss vorher der Treiber laufen, `nvidia-smi` zeigt dann die Karte an.

Schritt 2: Ein Modell laden und testen

```
ollama pull gemma4:12b-it-q4_K_M
ollama run gemma4:12b-it-q4_K_M "Fasse in drei Sätzen zusammen, wofür ein Lieferschein gut ist."
```

Mit `ollama ps` seht ihr, ob das Modell vollständig auf der Grafikkarte liegt. In der Spalte `PROCESSOR` sollte `100% GPU` stehen. Steht dort eine Aufteilung wie `48%/52% CPU/GPU`, ist das Modell zu groß für den Grafikspeicher und wird deutlich langsamer. Dann ein kleineres Modell oder eine stärkere Quantisierung wählen.

Unsere Wahl auf der RTX 3090: Für den Einstieg `gemma4:12b-it-q4_K_M` (8 GB). Es war in unserer Messung mit Abstand am schnellsten und lässt auf einer 24-GB-Karte viel Platz für längere Eingaben. Wer mehr Tiefe braucht, nimmt `gemma4:31b-it-q4_K_M` (20 GB). Es passt gerade noch ganz auf die Karte, sollte dort aber nicht mit sehr großem Kontext laufen (siehe Messung unten). Gemma 4 steht wie Qwen3.8 unter der Apache-Lizenz 2.0 und darf im Unternehmen ohne Einschränkung genutzt werden. Ältere Gemma-Fassungen hatten eigene Nutzungsbedingungen, dort lohnt ein Blick in die Lizenz.

Gemma 4 und Qwen3.8 denken vor der Antwort erst nach. Das verbessert schwierige Antworten, kostet aber Zeit und Ausgabetokens: In unserer Messung gingen bei Gemma 4 meist mehr als die Hälfte der erzeugten Tokens ins Nachdenken. Wer über die Schnittstelle arbeitet und kurze Antworten schnell braucht, kann das Denken mit `"think": false` abschalten.

Schritt 3: Ollama im Netzwerk freigeben

Ab Werk hört Ollama nur auf den eigenen Rechner (`127.0.0.1`, Port 11434). Läuft Open WebUI in Docker, reicht das nicht, auch wenn beides auf demselben Rechner steht: Der Container erreicht Ollama dann nicht. Dasselbe gilt, wenn ein anderer Rechner zugreifen soll. In beiden Fällen muss Ollama auf allen Netzwerkadressen hören:

- **Windows:** Ollama beenden, in den Umgebungsvariablen des Kontos `OLLAMA_HOST` mit dem Wert `0.0.0.0:11434` anlegen, Ollama neu starten.
- **Linux:** `sudo systemctl edit ollama`, unter `[Service]` die Zeile `Environment="OLLAMA_HOST=0.0.0.0:11434"` eintragen, dann `sudo systemctl daemon-reload` und `sudo systemctl restart ollama`.

Wichtig: Ollama selbst hat keine Anmeldung. Wer den Port erreicht, kann das Modell nutzen. Gebt den Port deshalb nur im internen Netz frei, nie ins Internet, und lasst die Benutzer über Open WebUI arbeiten, das eine Anmeldung hat. Unter Windows kann die Firewall eingehende Verbindungen auf 11434 und 3000 blockieren. Gebt beide Ports nur für das private Netzwerkprofil frei, nicht für öffentliche Netze.

Schritt 4: Open WebUI per Docker davorsetzen

Zuerst einen geheimen Schlüssel erzeugen und aufbewahren, zum Beispiel mit `openssl rand -hex 32`. Ohne festen Schlüssel sind nach jedem Update alle Nutzer abgemeldet.

```
docker run -d -p 3000:8080 --add-host=host.docker.internal:host-gateway \
  -v open-webui:/app/backend/data -e WEBUI_SECRET_KEY=EUER_SCHLUESSEL \
  --name open-webui --restart always ghcr.io/open-webui/open-webui:main
```

Open WebUI findet ein Ollama auf demselben Rechner von selbst. Danach im Browser `http://localhost:3000` öffnen, von anderen Rechnern `http://ADRESSE-DES-RECHNERS:3000`.

Ohne Docker geht es auch mit Python 3.11 oder 3.12: `pip install open-webui` und `open-webui serve`, dann läuft die Oberfläche auf Port 8080.

Schritt 5: Administrator und Teamkonten

Das **erste Konto**, das ihr in Open WebUI anlegt, wird Administrator. Danach ist die Registrierung automatisch geschlossen. Für das Team schaltet ihr über euer Profilbild unter *Einstellungen* → *Admin* → *Authentifizierung* neue Registrierungen frei (in der englischen Oberfläche *Settings* → *Admin* → *Authentication* → *New Sign Ups*). Neue Konten stehen dann auf „ausstehend“ (*Pending*), bis ihr sie im Adminbereich freigebt. Die deutschen Bezeichnungen können je nach Version leicht abweichen.

Was wir gemessen haben

Fünf Modelle auf der RTX 3090, jeweils Median aus vier deutschen Aufgaben mit je drei Läufen: eine kurze Frage, eine E-Mail, eine lange Betriebsanweisung mit rund 2.800 Tokens Eingabe und eine Frage zu sechs Auszügen aus Firmendokumenten. Temperatur 0, Kontext 8.192, höchstens 2.000 Ausgabetokens, Denkphase eingeschaltet, gemessen über die Ollama-Schnittstelle.

Modell	Datei	Auf der Grafikkarte	Ausgabe (Tokens/s)	Eingabe lesen (Tokens/s, lange Aufgabe)	Antwort fertig nach	Erster Start
`gemma4:12b-it-q4_K_M`	8,0 GB	ganz	151	2.605	5,8 s	8,6 s
`gemma4:12b-it-q8_0`	12,8 GB	ganz	54	3.085	13,5 s	11,1 s
`gemma4:31b-it-q4_K_M`	20,4 GB	ganz	82	1.030	9,3 s	17,6 s
`qwen3.8:27b-q4_K_M`	17,7 GB	ganz	43	1.505	8,5 s	13,6 s
`gemma4:26b-a4b-it-q8_0` (MoE)	28,1 GB	76 %, Rest im Arbeitsspeicher	55	1.712	14,6 s	24,4 s

„Antwort fertig nach“ ist die Zeit von der Anfrage bis zum letzten Token, einschließlich Denkphase, als Median über alle Aufgaben. „Erster Start“ ist das Laden des Modells von der Platte. Die Frage zu den Firmendokumenten haben alle fünf Modelle richtig beantwortet, mit den richtigen Quellenangaben.

Was die Zahlen heißen:

- **Alle fünf sind schnell genug für einen Chat.** Selbst das langsamste Modell schreibt deutlich schneller, als man mitliest. Was im Alltag auffällt, ist die Wartezeit bis zur fertigen Antwort, und die hängt stark davon ab, wie lange das Modell vorher nachdenkt. Qwen3.8 erzeugt die wenigsten Tokens je Antwort (im Median 313 gegen 570 bis 770 bei Gemma 4) und ist deshalb trotz der niedrigsten Tokenrate sogar etwas früher fertig als das große Gemma.

- **Zwei Gemma-Modelle sind schneller, als die Hardware allein erwarten lässt.** Die Speicherbandbreite der RTX 3090 erlaubt für ein dichtes 31B-Modell in Q4 normalerweise höchstens um die 45 Tokens pro Sekunde, wir haben 82 gemessen. Laut Modellinfo von Ollama sind die Tags `gemma4:12b-it-q4_K_M` und `gemma4:31b-it-q4_K_M` Fassungen mit Multi-Token-Vorhersage, die mehrere Tokens auf einmal erzeugen (siehe spekulatives Dekodieren). Für euch ist das ein Vorteil. Es heißt aber auch, dass der Abstand zwischen 12B in Q4 und in Q8 nicht allein von der Quantisierung kommt.
- **Auslagern geht, kostet aber beim Start.** Das MoE-Modell ist mit 28 GB größer als die Karte. Ollama legt rund ein Viertel in den Arbeitsspeicher (Offloading). Weil je Token nur ein kleiner Teil der Parameter rechnet, bleibt es mit 55 Tokens pro Sekunde brauchbar. Das Laden dauert aber fast dreimal so lang wie beim kleinen Modell.
- **Großer Kontext kann bremsen.** Die Ollama-App stand bei uns auf einem Kontext von 32.768 Tokens. Mit diesem Wert fiel das 31B-Modell von 82 auf rund 54 Tokens pro Sekunde. Bei den anderen Modellen änderte sich kaum etwas. Woran das genau liegt, haben wir nicht untersucht. Wer das 31B-Modell nutzt, sollte den Kontext nur so groß stellen, wie die Dokumente es wirklich brauchen.
- **Denkphase einplanen.** Mit einer Grenze von 300 Ausgabetokens, wie sie in manchen Messungen üblich ist, haben die Gemma-Modelle in unserem ersten Durchlauf nur nachgedacht und keine Antwort geliefert. Ein Grund mehr, das Ausgabelimit nicht zu knapp zu setzen oder das Denken bewusst abzuschalten.

Für den Betrieb im Team

- **Gleichzeitige Anfragen:** Ollama bearbeitet ab Werk eine Anfrage je Modell auf einmal (`OLLAMA_NUM_PARALLEL`). Für eine Handvoll Nutzer reicht das meist, für eine ganze Abteilung lohnt ein Server wie vLLM.
- **Kontextlänge:** Ab Werk sieht das Modell 4.096 Tokens. Für längere Dokumente `OLLAMA_CONTEXT_LENGTH` erhöhen, das kostet Grafikspeicher. Die Windows-App hat dafür zusätzlich eine eigene Einstellung, bei uns stand sie auf 32.768. Schaut nach, was dort eingestellt ist, bevor ihr euch über langsame Antworten wundert.
- **Modell geladen halten:** Ollama entlädt ein Modell nach fünf Minuten ohne Anfrage. Wer Wartezeiten beim ersten Aufruf vermeiden will, setzt `OLLAMA_KEEP_ALIVE` höher.
- **Updates:** Ollama aktualisiert sich unter Windows selbst, unter Linux das Installationsskript erneut ausführen. Open WebUI: `docker pull ghcr.io/open-webui/open-webui:main`, Container neu anlegen, mit demselben `WEBUI_SECRET_KEY`.
- **Lizenz:** Open WebUI steht seit Version 0.6.6 (April 2025) unter einer eigenen Lizenz. Intern nutzen darf es jedes Unternehmen mit beliebig vielen Nutzern. Das Open-WebUI-Logo zu entfernen oder durch das eigene zu ersetzen, ist ab mehr als 50 Nutzern in 30 Tagen nur mit einer Enterprise-Lizenz erlaubt. Auch die Lizenz des gewählten Modells prüfen, siehe offene Gewichte.
- **Datenschutz:** Lokal heißt, dass die Texte das Haus nicht verlassen. Trotzdem speichert Open WebUI Gesprächsverläufe; legt fest, wer sie sehen darf und wann sie gelöscht werden. Mehr dazu in DSGVO-konforme LLMs 2026.

Quellen

- Ollama-Dokumentation: Windows, Linux, FAQ, API, abgerufen am 24.10.2026
- Open-WebUI-Dokumentation: Schnellstart, Lizenz, abgerufen am 24.10.2026, Schnellstart erneut am 25.10.2026
- Messwerte: eigene Messung am 25.10.2026 auf AMD Ryzen 9 5900X, 64 GB, NVIDIA RTX 3090 (Treiber 610.88), Windows 11 Pro, Ollama 0.40.11; Testaufgaben und Rohdaten auf Anfrage
- Modelle und Lizenzen: Ollama-Bibliothek Gemma 4 und Qwen3.8, Gemma 4 auf Hugging Face, Qwen3.8-27B auf Hugging Face, abgerufen am 25.10.2026

Passend dazu

Insights

Ubuntu 26.10 und Myna: Diktieren mit lokaler KI, was Betriebe vor dem Einschalten wissen sollten

· Insights

Ollama im Unternehmen: KI-Modelle auf dem eigenen Server, gemessen ohne Grafikkarte

· Wissen

Fine-Tuning oder RAG: wie bringt man Firmenwissen in ein KI-Modell?

· Insights

Whisper im Unternehmen: Spracherkennung auf dem eigenen Server, gemessen ohne Grafikkarte

Konkrete Frage zu eurem Setup?

Ein 30-Minuten-Erstgespräch klärt meistens schon, ob euer aktueller AI-Stack hält oder wo nachzuarbeiten ist. Kostenlos, ohne Verkaufsdruck.