Mittelstand-Digital
Zentrum
Fokus Mensch

Admin-Manual und Lessons Learned

DEPLOYMENT EINES LOKALEN KI-AGENTEN AM BEISPIEL VON KARL (KI-ASSISTENT RECHNET LOKAL)

Ein Demonstrator des Mittelstand-Digital Zentrums Fokus Mensch

Stand: 19. August 2026

Inhaltsverzeichnis

Ziel des Demonstrators und dieses Manuals

Dieses Manual zeigt, wie ein digital souveräner KI-Agent aufgebaut wird, der vollständig lokal läuft. Der Demonstrator KARL (KI-Assistent Rechnet Lokal) nimmt seinen Nutzerinnen und Nutzern übliche Büro- und Rechercheaufgaben ab, unterstützt beim Programmieren, hilft bei administrativen IT-Tätigkeiten und lässt sich um weitere Fähigkeiten erweitern. Alle Daten und alle Rechenschritte bleiben dabei auf der eigenen Hardware. Es fließen keine Daten in die Cloud, es fallen keine laufenden API-Kosten an und das System funktioniert auch ohne Internetverbindung.

Drei Bausteine machen das möglich. Als Agenten-Framework nutzen wir den quelloffenen Hermes-Agent. Die nötige Rechenleistung liefern zwei per Thunderbolt verbundene Macbook Pro M5 Max, auf denen wir verschiedene quelloffene Sprachmodelle ausprobiert haben (Qwen und MiniMax in unterschiedlichen Größen), die der Agent für seine Arbeit nutzt. Den Zugang schafft eine Anbindung an den Messenger Telegram, über die sich das agentische System vom Smartphone oder PC aus ortsunabhängig bedienen lässt.

Wichtige Sicherheitshinweise

Ein KI-Agent, der selbstständig Werkzeuge nutzt, ist mehr als ein Chatbot. Software dieser Art kann Befehle auf dem Rechner ausführen, Dateien lesen und schreiben und mit dem Internet kommunizieren. Genau daraus entstehen reale Risiken, von unbeabsichtigten Änderungen am System bis zum Abfluss vertraulicher Daten. Eine besondere Rolle spielt die sogenannte Prompt Injection: Manipulierte Inhalte, etwa auf einer Webseite oder in einem Dokument, können einem Agenten unbemerkt Anweisungen unterschieben.

Wir empfehlen deshalb ausdrücklich, solche Systeme zunächst in einer vom eigenen Arbeitsrechner getrennten Umgebung auszuprobieren, etwa auf einem dedizierten Testgerät oder in einer abgeschotteten Sandbox. Für den produktiven Einsatz gilt: Nur wer sich mit Netzwerk- und IT-Sicherheit auskennt, sollte agentische Systeme dauerhaft betreiben. Die folgenden Grundregeln haben sich dabei als Minimum etabliert:

- **Zugriff strikt begrenzen.** Der Telegram-Zugang gehört ausschließlich auf eine Freigabeliste definierter Nutzerkennungen, niemals offen für alle. Bietet das Framework ein Pairing-Verfahren mit Einmalcodes an, ist dieses der harten Verdrahtung vorzuziehen.
- **Rechte minimal halten.** Der Agent braucht kein Administratorkonto. Gefährliche Befehle sollten eine ausdrückliche Bestätigung erfordern. Viele Frameworks bringen dafür Freigabe- und Sperrlisten mit, die man aktiv pflegen sollte.
- **Ausführung isolieren.** Befehle des Agenten möglichst in einer Sandbox oder einem Container ausführen lassen, getrennt vom restlichen System und von persönlichen Daten.

- **Netzwerk abschotten.** Die Cluster-API und die Oberflächen des Agenten nur auf dem Gerät selbst oder im lokalen Netz erreichbar machen und niemals ungeschützt ins Internet stellen.
- **Zugangsdaten schützen.** Bot-Token, Schlüssel und Konfigurationsdateien mit restriktiven Dateirechten versehen und nicht in geteilten Ordnern oder Repositories ablegen.
- **Fremde Inhalte als Risiko behandeln.** Webseiten, E-Mails und Dokumente, die der Agent verarbeitet, können Prompt-Injection-Angriffe enthalten. Werkzeuge mit Internetzugriff deshalb bewusst und sparsam freischalten.
- **Betrieb überwachen.** Protokolle regelmäßig prüfen, Software aktuell halten und nach jeder Konfigurationsänderung die Sicherheitseinstellungen erneut kontrollieren.

Beide im Umfeld dieses Demonstrators relevanten Projekte pflegen ausführliche eigene Sicherheitsleitfäden, deren Lektüre wir vor der Inbetriebnahme empfehlen: die Sicherheitsdokumentation des Hermes-Agent (github.com/NousResearch/hermes-agent, Nutzerhandbuch, Abschnitt Security) sowie der Sicherheitsleitfaden des verwandten Open-Source-Agenten OpenClaw (docs.openclaw.ai, Abschnitt Gateway Security), der Bedrohungsmodell und Schutzmaßnahmen agentischer Systeme besonders anschaulich beschreibt.

Über dieses Dokument

Dieses Dokument richtet sich an IT-Verantwortliche in kleinen und mittleren Unternehmen. Es beschreibt, wie das System aufgebaut ist, wie es sich mit zwei Macbooks nachbauen lässt und worauf es bei der Konfiguration ankommt. Die Lessons Learned am Ende sind bewusst ausführlich geraten. Einiges hat bei uns im ersten Anlauf nicht funktioniert, und genau diese Erfahrungen sparen beim Nachbau die meiste Zeit.

Ein Hinweis zur Entstehung: Aufbau und Administration des Clusters haben wir weitgehend mit einem KI-Coding-Agenten (Claude Code) durchgeführt, der die Kommandos auf den Geräten ausgeführt und dokumentiert hat. Dieses Manual fasst die Protokolle aus rund sechs Wochen Aufbau und Betrieb zusammen (Juli bis August 2026).

1. Das System im Überblick

Die Grundidee: Zwei baugleiche Macbooks mit Apple Silicon bündeln ihren gemeinsamen Arbeitsspeicher, sodass ein Sprachmodell laufen kann, das für ein einzelnes Gerät zu groß wäre. Apple stellt dafür mit MLX ein quelloffenes Framework bereit, das verteilte Inferenz über eine direkte Thunderbolt-Verbindung unterstützt. Auf dem Cluster läuft ein OpenAI-kompatibler API-Server, an den sich beliebige Anwendungen anbinden lassen. In unserem Fall ist das der quelloffene Hermes-Agent von Nous Research, der unter anderem eine Telegram-Anbindung mitbringt.

Eine Nachricht nimmt also folgenden Weg: Telegram, Hermes-Agent, lokale API des MLX-Clusters, verteiltes Sprachmodell auf beiden Macs, und die Antwort denselben Weg zurück.

Baustein	Aufgabe	Eingesetzte Lösung
Hardware	Rechenleistung und Speicher	Zwei baugleiche MacBook Pro (M5 Max), je 128 GB gemeinsamer Speicher
Verbindung	Datenaustausch zwischen den Geräten	Thunderbolt-Kabel, Thunderbolt-Bridge mit festen IP-Adressen, RDMA
Inferenz	Verteilter Betrieb des Sprachmodells	MLX und mlx-lm (Apple, Open Source) mit JACCL-Backend
Sprachmodell	Sprachverständnis und Tool-Aufrufe	Zunächst MiniMax M2.5 (4-Bit), inzwischen Qwen 3.8 27B (4-Bit) im verteilten Betrieb
Schnittstelle	Zugang für Anwendungen	OpenAI-kompatible API, bei uns auf Port 8091
Agent	Aufgaben ausführen, Werkzeuge nutzen	Hermes-Agent (Nous Research, Open Source)
Bedienung	Zugang für Nutzerinnen und Nutzer	Telegram-Bot, dazu Desktop-App und Web-Dashboard des Agenten

Zur Begrifflichkeit: Der erste Mac koordiniert den Verbund und stellt die API bereit, wir nennen ihn im Folgenden Master. Der zweite Mac steuert Speicher und Rechenleistung bei, wir nennen ihn Slave.

2. Voraussetzungen

- Zwei Macs mit Apple Silicon, möglichst baugleich und mit großzügigem Speicherausbau. In unserem Demonstrator sind es zwei identische MacBook Pro mit M5 Max und je 128 GB gemeinsamem Speicher.
- Ein aktuelles macOS auf beiden Geräten. Für RDMA über Thunderbolt wird macOS 26 oder neuer benötigt.
- Ein Thunderbolt-Kabel, möglichst Thunderbolt 4 oder 5.
- Ausreichend Festplattenplatz. Die Modelldateien liegen auf beiden Geräten und belegen je nach Modell 30 bis 130 GB pro Gerät.
- Ein schneller Internetanschluss für die einmaligen Modell-Downloads.
- Ein Telegram-Konto für die Bot-Einrichtung.
- Grundkenntnisse im Umgang mit Terminal und SSH.

Der reine Grundaufbau ist an einem Arbeitstag zu schaffen. Für Feinschliff, Tests und Stabilisierung sollte man erfahrungsgemäß deutlich mehr Zeit einplanen. Bei uns lagen zwischen dem ersten Aufbau und dem stabilen Dauerbetrieb mehrere Wochen mit einzelnen Wartungseinsätzen.

3. Aufbau Schritt für Schritt

3.1 Beide Macs vorbereiten

Der Cluster sollte möglichst auf dedizierten Geräten laufen. Alles, was nennenswert Arbeitsspeicher belegt, konkurriert später direkt mit dem Sprachmodell (mehr dazu in den Lessons Learned). Bei uns liefen auf dem Master anfangs mehrere Docker-Container und lokale KI-Dienste, die wir für den Clusterbetrieb gestoppt und aus dem Speicher entladen haben.

Anschließend auf beiden Geräten die entfernte Anmeldung aktivieren: Systemeinstellungen, Allgemein, Teilen, Entfernte Anmeldung. Der Terminalbefehl sudo systemsetup -setremotelogin on scheitert ohne Festplattenvollzugriff des Terminals, der Weg über die Systemeinstellungen ist zuverlässiger.

Danach einen SSH-Schlüssel auf dem Master erzeugen und auf den Slave übertragen:

```
ssh-keygen -t ed25519
ssh-copy-id nutzer@<ip-oder-name-des-slaves>
ssh nutzer@<ip-oder-name-des-slaves>   # Test ohne Passwort
```

Zwei Einstellungen, die im Dauerbetrieb wichtig werden: den Ruhezustand auf beiden Geräten deaktivieren und die automatische Anmeldung aktivieren. Ein Mac, der am Anmeldebildschirm steht oder schläft, nimmt am Cluster nicht teil. Genau daran sind bei uns anfangs mehrere Tests gescheitert.

3.2 Thunderbolt-Verbindung einrichten

Die beiden Macs mit dem Thunderbolt-Kabel verbinden. In den Systemeinstellungen unter Netzwerk auf beiden Geräten einen Dienst vom Typ Thunderbolt-Bridge hinzufügen und feste IP-Adressen vergeben, bei uns 10.0.0.1 für den Master und 10.0.0.2 für den Slave, jeweils mit der Subnetzmaske 255.255.255.0.

Auch hier gilt: Der Weg über die grafischen Systemeinstellungen ist der verlässlichste. Unsere Versuche, den Netzwerkdienst per networksetup im Terminal anzulegen, endeten wiederholt mit der Fehlermeldung, die Systemkonfigurationsdatenbank sei nicht erreichbar. Über die Oberfläche funktionierte dieselbe Einstellung sofort.

Damit die Adressen einen Neustart überleben, haben wir auf beiden Geräten ein kleines Skript als LaunchDaemon hinterlegt, das die IP beim Systemstart setzt. Beispiel für den Master:

```
# /usr/local/bin/tb-ip-setup.sh
#!/bin/sh
/usr/sbin/networksetup -setmanual "Thunderbolt Bridge" \
    10.0.0.1 255.255.255.0

sudo install -m 755 tb-ip-setup.sh /usr/local/bin/
sudo install -m 644 de.beispiel.tb-cluster-ip.plist /Library/LaunchDaemons/
sudo launchctl bootstrap system \
    /Library/LaunchDaemons/de.beispiel.tb-cluster-ip.plist
```

Auf dem Slave das Gleiche mit der Adresse 10.0.0.2. Zum Abschluss die Verbindung testen, etwa mit ping 10.0.0.2 vom Master aus.

3.3 RDMA aktivieren

RDMA (Remote Direct Memory Access) erlaubt den beiden Geräten, direkt in den Speicher des jeweils anderen zu schreiben. Das senkt die Latenz der Thunderbolt-Verbindung deutlich und ist für flüssige verteilte Inferenz praktisch Pflicht. Aktiviert wird es auf beiden Macs im Terminal, danach ist jeweils ein Neustart erforderlich:

```
sudo rdma_ctl enable    # aktivieren, danach Neustart
sudo rdma_ctl disable   # bei Bedarf wieder abschalten
```

Die Einstellung übersteht normale Neustarts und Updates. Wer einen der Macs später vollständig zurücksetzt, sollte sie danach erneut setzen. Wichtig für die Prüfung nach dem Neustart: Beide Geräte müssen tatsächlich angemeldet sein, ein Gerät am Anmeldebildschirm meldet den RDMA-Status nicht zuverlässig.

3.4 MLX installieren

Auf beiden Geräten wird eine Python-Umgebung mit MLX und mlx-lm benötigt. Wir haben dafür den Paketmanager uv verwendet und die Umgebung in einem eigenen Verzeichnis angelegt, auf dem Slave per SSH auf identische Weise:

```
curl -LsSf https://astral.sh/uv/install.sh | sh
mkdir -p ~/mlx-cluster && cd ~/mlx-cluster
uv venv && source .venv/bin/activate
uv pip install mlx mlx-lm
```

Wichtig ist, dass die Versionen von Python, MLX und mlx-lm auf beiden Geräten übereinstimmen. Der Verbund reagiert empfindlich auf Versionsunterschiede.

3.5 Modell auswählen und herunterladen

Die Modellwahl ist die wichtigste Einzelentscheidung des ganzen Projekts, Kapitel 4 behandelt sie ausführlich. Kurz zusammengefasst muss ein Kandidat drei Fragen bestehen: Passt er mit deutlich Luft nach oben in den gemeinsamen Speicher? Lässt er sich mit MLX über zwei Geräte verteilen? Beherrscht er zuverlässig Tool-Aufrufe für den Agentenbetrieb?

Fertig quantisierte MLX-Varianten gängiger Modelle stellt die Community auf Hugging Face bereit (Organisation mlx-community). Der Download erfolgt über das Kommandozeilenwerkzeug von Hugging Face. Zwei Praxis-Tipps aus unseren Downloads von über 120 GB:

- Das Modell nur einmal auf dem Master herunterladen und dann über die Thunderbolt-Verbindung auf den Slave spiegeln, zum Beispiel mit rsync. Das ist um ein Vielfaches schneller als ein zweiter Download.
- Bei wiederholt abbrechenden Downloads das xet-Backend des Hugging-Face-Clients deaktivieren (Umgebungsvariable HF_HUB_DISABLE_XET=1). Bei uns brachen große Downloads mit xet mehrfach ab und liefen ohne xet stabil durch.

```
export HF_HUB_DISABLE_XET=1
hf download mlx-community/<modellname> \
    --local-dir ~/mlx-cluster/models/<modellname>
rsync -avP ~/mlx-cluster/models/ \
    nutzer@10.0.0.2:~/mlx-cluster/models/
```

Während langer Downloads und Kopiervorgänge verhindert der Befehl caffeinate, dass eines der Geräte einschläft.

3.6 Cluster starten und testen

Der verteilte Start erfolgt über den Launcher von MLX, der das Modell auf beide Geräte lädt und auf dem Master einen OpenAI-kompatiblen API-Server bereitstellt. Der Aufruf sieht sinngemäß so aus, die genauen Parameter hängen von der eingesetzten mlx-lm-Version ab und sollten in der aktuellen Dokumentation geprüft werden:

```
mlx.launch --backend jaccl --hosts 10.0.0.1,10.0.0.2 \
    -- mlx_lm.server \
    --model ~/mlx-cluster/models/<modellname> \
    --host 0.0.0.0 --port 8091 --max-kv-size 65536
```

Zwei Punkte verdienen besondere Aufmerksamkeit. Erstens sollte das Kontextfenster fest begrenzt werden, bei uns auf 65.536 Token, statt das Maximum des Modells zu übernehmen. Die Begründung liefert Kapitel 4.1. Zweitens braucht der erste Ladevorgang Geduld, das Modell wird dabei in den Speicher beider Geräte verteilt. Je nach Modellgröße können mehrere Minuten vergehen, bevor die erste Antwort kommt.

Anschließend die API mit einer Testanfrage prüfen:

```
curl http://10.0.0.1:8091/v1/chat/completions \
  -H "Content-Type: application/json" \
  -d '{"model": "<modellname>",
       "messages": [{"role": "user",
                     "content": "Bitte kurz antworten."}]}'
```

3.7 Hermes-Agent installieren und anbinden

Als Agent setzen wir den quelloffenen Hermes-Agent von Nous Research ein (github.com/NousResearch/hermes-agent). Er bringt neben der Kommandozeile eine Desktop-App, ein Web-Dashboard und fertige Anbindungen an Messenger wie Telegram mit. Wir haben ihn nativ auf dem Master installiert, also bewusst ohne Docker. Das vermeidet Container-Overhead im ohnehin knappen Speicher und gibt dem Agenten direkten Zugriff auf das System, was für seine Werkzeuge nötig ist.

Nach der Installation wird der Agent auf den lokalen Endpunkt des Clusters konfiguriert, also auf http://10.0.0.1:8091/v1 mit dem Namen des geladenen Modells. Eine Eigenheit, die uns Zeit gekostet hat: Die Modellauswahl in den Oberflächen solcher Werkzeuge zeigt oft nur vorkonfigurierte Kataloge an. Der eigene Cluster-Endpunkt musste bei uns von Hand als Eintrag ergänzt werden, ein Klick auf Aktualisieren genügte nicht.

Damit der Agent einen Neustart des Geräts übersteht, haben wir ihn als LaunchAgent in den Autostart gelegt. Wer das System zeitweise ohne Internetzugang betreibt, findet im Log regelmäßig Warnungen, weil der Agent Modellkataloge im Internet abfragen möchte. Diese Meldungen sind harmlos, die lokale Funktion ist nicht beeinträchtigt.

3.8 Telegram anbinden

Für den Telegram-Zugang wird zunächst über den offiziellen Telegram-Bot BotFather ein eigener Bot angelegt. Der dabei erzeugte Token kommt in die Konfiguration des Hermes-Agenten. Anschließend braucht der Agent die numerische Telegram-Nutzerkennung der Personen, die ihn verwenden dürfen. Die eigene Kennung lässt sich leicht ermitteln, indem man dem neuen Bot eine Nachricht schickt und die Antwort der Bot-API abruft, dort steht sie im Feld id unterhalb von from.

Diese Kennung wird im Agenten als Home Channel eingetragen, an den der Agent auch von sich aus Ergebnisse und Benachrichtigungen schickt. Ein Punkt ist uns dabei besonders wichtig: Der Zugriff muss zwingend auf definierte Nutzerkennungen beschränkt werden. Der Agent kann Werkzeuge auf dem Master ausführen, ein offener Bot wäre ein erhebliches Sicherheitsrisiko.

Danach ist KARL einsatzbereit: Eine Nachricht an den Bot wird vom lokalen Modell auf dem Cluster beantwortet, egal wo man sich gerade befindet.

4. Konfiguration richtig dimensionieren

Dieses Kapitel bündelt die Erfahrungswerte, die über Erfolg und Misserfolg des Aufbaus entschieden haben. Wer hier sauber plant, erspart sich die meisten unserer Umwege.

4.1 Das Speicherbudget: Modell plus Kontext plus System

Die zentrale Rechnung lautet: Modellgewichte plus KV-Cache plus Betriebssystem und Anwendungen müssen zusammen unter dem real verfügbaren Speicher bleiben. Der KV-Cache ist der Arbeitsspeicher des Gesprächs, er wächst mit jedem Token im Kontextfenster. Genau dieser Posten wird gern unterschätzt, besonders bei agentischen Anwendungen: Systemprompts, Werkzeugbeschreibungen, Werkzeugausgaben und Zwischenergebnisse füllen das Kontextfenster erheblich schneller, als es ein normaler Chat je würde.

Unser Lehrstück: MiniMax M2.5 belegte in der 4-Bit-Quantisierung bereits rund 123 GB, verteilt auf zwei Geräte mit zusammen 256 GB. Auf dem Papier passte das. Im Betrieb blieb aber so wenig Luft für den KV-Cache, dass der Verbund abstürzte, sobald der Kontext im Agentenbetrieb anwuchs. Zeitweise riss es das ganze System mit, der Master war mehrfach komplett eingefroren.

Das größte Modell, das gerade noch passt, ist die falsche Wahl. Das richtige Modell lässt Platz zum Arbeiten.

Als Orientierung hat sich bewährt: Die Modellgewichte sollten höchstens die Hälfte bis zwei Drittel des Gesamtspeichers belegen, der Rest gehört dem Kontext und dem System. Zusätzlich sollte das Kontextfenster beim Start fest begrenzt werden, statt das Maximum des Modells zu übernehmen. Bei uns haben sich 65.536 Token als tragfähiger Kompromiss zwischen agentischer Arbeitsfähigkeit und Stabilität erwiesen. Ein Lasttest mit rund 30.000 Token Kontext lief mit dem aktuellen Modell ohne Auffälligkeiten.

4.2 Ein toolcall-fähiges Modell wählen

Ein Agent ist nur so gut wie die Fähigkeit seines Modells, Werkzeuge korrekt aufzurufen. Das Modell muss strukturierte Funktionsaufrufe (Tool Calling beziehungsweise Function Calling) zuverlässig beherrschen und dafür trainiert sein. In der Praxis heißt das: eine Instruct-Variante wählen, die Modellkarte auf explizite Tool-Calling-Unterstützung prüfen und einschlägige Vergleiche heranziehen, etwa öffentliche Function-Calling-Benchmarks oder die Rankings von Artificial Analysis. Ein Modell, das im Chat brilliert, aber Werkzeugaufrufe nur unsauber formatiert, macht den Agenten im Alltag unbrauchbar.

4.3 Verteilbarkeit vorab prüfen

Nicht jedes Modell lässt sich mit MLX über zwei Rechner verteilen. Die Architektur muss vom Framework unterstützt werden, und die Unterstützung reift je Modellfamilie unterschiedlich schnell. Wir haben das auf die harte Tour gelernt: Qwen3 235B ließ sich erst mit Anpassungen am Code überhaupt verteilt starten und blieb dann beim Laden hängen, auf dem Master standen nach Minuten 18 von 37 GB im Speicher, auf dem Slave 1 von 86 GB. Diesen Versuch haben wir nach mehreren Anläufen aufgegeben. MiniMax M2.5 und später Qwen 3.6 27B liefen dagegen ohne Anpassungen.

Die Empfehlung daher: Vor jedem großen Download gezielt suchen, ob es Berichte über erfolgreichen verteilten Betrieb genau dieses Modells mit mlx-lm gibt. Eine Viertelstunde Recherche erspart einen Download von 125 GB für die Tonne.

4.4 Quantisierung mit Augenmaß

Quantisierung reduziert den Speicherbedarf eines Modells, indem die Gewichte mit geringerer Genauigkeit gespeichert werden. 4-Bit ist der übliche Kompromiss aus Größe und Qualität. Eine 3-Bit-Variante spart weiter Speicher, kostet aber spürbar Antwortqualität. Wir haben die 3-Bit-Variante unseres damaligen Modells als Notlösung gegen die Speichernot getestet und uns am Ende bewusst dagegen entschieden. Die bessere Antwort auf Speichernot ist ein kleineres Modell in guter Quantisierung, nicht ein großes in schlechter.

Ein Wort zu Mixture-of-Experts-Modellen (MoE), zu denen sowohl MiniMax M2.5 als auch die großen Qwen-Modelle gehören: Pro Token rechnet nur ein Teil des Modells mit, das macht sie angenehm schnell. Für das Speicherbudget zählt aber trotzdem die volle Modellgröße, denn alle Experten müssen geladen sein.

4.5 Größe schlägt nicht Stabilität

Unsere Modellreihenfolge erzählt die Geschichte am besten: Qwen3 235B scheiterte an der Verteilbarkeit. MiniMax M2.5 mit rund 230 Milliarden Parametern lief und lieferte gute Antworten, riss den Verbund aber bei wachsendem Kontext regelmäßig in den Absturz. Qwen 3.6 27B schließlich läuft stabil, lässt reichlich Platz für ein großes Kontextfenster und beantwortet die Aufgaben des Demonstrators zuverlässig. Der Qualitätsverlust im Alltag ist deutlich kleiner, als es der Größenunterschied vermuten lässt, aktuelle kompakte Modelle überholen in den einschlägigen Rankings regelmäßig die großen Modelle der Vorgeneration.

Für den Agentenbetrieb ebenfalls hilfreich: Bei Modellen mit zuschaltbarem Reasoning lässt sich der Denkaufwand konfigurieren. Wir haben ihn auf die mittlere Stufe gesetzt. Das spart Token, Zeit und Speicher, ohne die Ergebnisqualität im Alltag sichtbar zu drücken. Inzwischen setzen wir mit Qwen 3.8 27B in 4-Bit-Quantisierung auf die Nachfolgeversion des Modells.

4.6 Ein Mac oder zwei? Der Geschwindigkeitsvergleich

Ein kompaktes Modell wie das von uns eingesetzte Qwen 3.8 27B in 4-Bit-Quantisierung wirft eine berechtigte Frage auf: Wozu dann noch ein Cluster? Mit rund 16 GB passt das Modell bequem auf einen einzelnen Mac. Der Theorie nach spricht sogar einiges für den Einzelbetrieb, denn das Modell findet auf einem Gerät reichlich Platz und die gesamte Abstimmung zwischen den beiden Rechnern entfällt, die im verteilten Betrieb bei jedem einzelnen Token über das Kabel läuft.

Die Praxis hat uns eines Besseren belehrt. Wir haben beide Varianten unter identischen Bedingungen gemessen: dasselbe Modell (Qwen 3.8 27B, 4-Bit), derselbe Prompt, jeweils 350 erzeugte Token, zwei Messläufe pro Variante nach einem Aufwärmlauf, gemessen über die API. Im Einzelbetrieb auf einem Mac lieferte das System rund 28 Token pro Sekunde. Verteilt über beide Macs waren es rund 43 Token pro Sekunde, ein Plus von etwa 54 Prozent bei kaum streuenden Messwerten.

Betriebsart	Lauf 1	Lauf 2	Durchschnitt
Einzelbetrieb (ein Mac)	27,4 Token/s	28,5 Token/s	rund 28 Token/s
Verteilt (zwei Macs)	43,5 Token/s	42,9 Token/s	rund 43 Token/s

Die Erklärung liegt in der Natur der Texterzeugung. Für jedes erzeugte Token muss das System sämtliche Modellgewichte aus dem Speicher lesen, das Tempo wird deshalb vor allem von der Speicherbandbreite begrenzt. Im verteilten Betrieb hält jeder Mac nur die

Hälfte der Gewichte und muss entsprechend nur die halbe Datenmenge lesen. Dieser Gewinn überwiegt den Mehraufwand für die Abstimmung über das Thunderbolt-Kabel deutlich, aus der theoretisch möglichen Verdopplung wird in der Praxis das gemessene Plus von gut der Hälfte.

Ein Selbstläufer ist die Verteilung trotzdem nicht. Der Einzelbetrieb ist robuster, weil nur ein Gerät laufen muss und die RDMA-Verbindung keine Rolle spielt. Der verteilte Betrieb ist schneller, verlangt aber, dass beide Macs wach und sauber verbunden sind. Wie diese Rechnung auf anderer Hardware ausgeht, hängt von Speicherbandbreite, Verbindungsqualität und Modellgröße ab. Unsere Empfehlung lautet deshalb: Probieren Sie mit der vorhandenen Hardware selbst aus, was möglich und was besser ist. Der Vergleich kostet wenige Minuten und ersetzt Bauchgefühl durch eine belastbare Entscheidungsgrundlage. Wir haben uns auf Basis der Messwerte für den verteilten Betrieb als Standard entschieden, der Einzelbetrieb bleibt als Rückfallebene eine Kommandozeile entfernt.

Ausblick: Beim Tempo ist das letzte Wort noch nicht gesprochen. In der MLX-Community läuft derzeit eine offene Challenge mit dem Ziel, den Token-Durchsatz genau dieses Modells (Qwen 3.8 27B) auf Apple Silicon zu maximieren, also die Inferenz noch effizienter zu machen. Der Hebel dafür ist vor allem Multi-Token-Prediction mit spekulativem Dekodieren. Der aktuelle Spitzenwert liegt bei über 85 Token pro Sekunde auf einem einzelnen Gerät und damit beim gut Dreifachen der Ausgangsbasis. Solche Fortschritte fließen erfahrungsgemäß in die frei verfügbaren Werkzeuge zurück, es lohnt sich also, die Entwicklung zu verfolgen: www.yukon.org/mlxfast.

5. Betrieb und Wartung

- **Autostart durchgängig einrichten.** Feste Thunderbolt-IPs als LaunchDaemon, der Agent als LaunchAgent, der Cluster-Start über ein Skript. Ziel ist, dass der Verbund nach einem Stromausfall ohne Handarbeit wieder hochkommt.
- **Nach jedem Neustart eine feste Prüfroutine fahren:** Sind beide Geräte angemeldet und wach? Steht die Thunderbolt-Verbindung (ping)? Ist RDMA aktiv? Antwortet die Cluster-API auf eine Testanfrage?
- **Eine Monitoring-Eigenheit kennen:** Übliche Auslastungsanzeigen melden auf beiden Macs dauerhaft 100 Prozent GPU-Last, auch wenn der Cluster nichts tut. Das ist eine Eigenheit des verteilten MLX-Betriebs, die Prozesse warten aktiv aufeinander. Lüfter, Temperatur und Energieverbrauch sind die ehrlicheren Indikatoren.
- **Versionen synchron halten.** macOS, Python, MLX und mlx-lm sollten auf beiden Geräten immer im Gleichstand sein. Updates daher stets paarweise einspielen.
- **Störungen pragmatisch beheben.** Geht der Slave verloren (Kabel gezogen, Neustart, Ruhezustand), ist der schnellste Weg fast immer: Cluster-Prozesse auf beiden Seiten beenden, Verbindung prüfen, neu starten, Testanfrage schicken.

6. Stromverbrauch und Betriebskosten

Für die Bewertung eines lokalen KI-Systems lohnt der Blick auf die Gesamtkosten über die Nutzungsdauer (Total Cost of Ownership). Die Rechnung ist angenehm kurz. Den größten Posten bildet die einmalige Anschaffung der beiden Rechner. Dazu kommen geringe Kosten für die Infrastruktur, also das Thunderbolt-Kabel, das Netzwerk und der ohnehin vorhandene Internetanschluss. Im laufenden Betrieb bleibt danach nur noch ein Posten übrig: Strom. Nutzungsabhängige Gebühren, wie sie bei Cloud-Diensten je Anfrage oder je Token anfallen, entstehen nicht.

Den Verbrauch messen wir mit einer einfachen Messsteckdose, an der beide Macs gemeinsam angeschlossen sind. Das Ergebnis: Je nach Auslastung pendelt der Verbund zwischen rund 50 Watt im Leerlauf und etwa 160 Watt unter Volllast, gemessen für beide Geräte zusammen. Die folgende Momentaufnahme aus dem laufenden Betrieb zeigt gut 70 Watt. Für ein System, das ein großes Sprachmodell bereithält, sind das niedrige Werte, ein einzelner Arbeitsplatzrechner mit dedizierter Grafikkarte liegt unter Last deutlich darüber.

Messung mit einer Zwischenstecker-App: aktuelle Leistungsaufnahme des Clusters mit beiden Macs im laufenden Betrieb (Momentaufnahme: 71,71 Watt)

Eine Beispielrechnung zur Einordnung: Läuft der Cluster rund um die Uhr bei durchschnittlich 100 Watt, ergibt das etwa 2,4 Kilowattstunden am Tag und rund 880 Kilowattstunden im Jahr. Bei einem Strompreis von 30 Cent je Kilowattstunde entspricht das ungefähr 260 Euro im Jahr. Im überwiegenden Leerlauf halbiert sich dieser Wert etwa, unter dauerhafter Volllast steigt er auf grob 420 Euro. Selbst im ungünstigsten Fall bleiben die laufenden Kosten damit überschaubar und vor allem planbar. Eine Messsteckdose für kleines Geld macht die eigenen Werte sichtbar und gehört deshalb zu unserer Empfehlung für den Betrieb.

7. Lessons Learned

Zum Abschluss die wichtigsten Erfahrungen aus sechs Wochen Aufbau und Betrieb, jeweils mit dem, was wir heute anders machen würden.

Zu groß gedacht ist schnell gescheitert

Der Reflex, das größtmögliche Modell zu wollen, ist verständlich und war unser teuerster Fehler. Speicherwarnungen, eingefrorene Geräte und Abstürze bei wachsendem Kontext haben uns über Wochen begleitet, bis wir das Modell deutlich verkleinert haben. Heute würden wir von Anfang an mit einem kompakten Modell starten, den stabilen Betrieb beweisen und erst dann schrittweise vergrößern.

Verteilbarkeit ist keine Selbstverständlichkeit

Dass sich ein Modell technisch herunterladen lässt, heißt nicht, dass es verteilt läuft. Der gescheiterte Anlauf mit Qwen3 235B inklusive Code-Anpassungen und hängendem Ladevorgang hat uns einen Abend und 125 GB Download gekostet. Erst recherchieren, dann herunterladen.

Der Cluster teilt seinen Rechner ungern

Auf dem Master liefen anfangs Docker-Container mit weiteren Diensten, ein lokaler Ollama-Dienst mit dauerhaft geladenem Modell und LM Studio parallel. Die Folge waren Speicherwarnungen mitten im Modellstart. Wir haben schließlich alle konkurrierenden Dienste gestoppt und aus dem Speicher entladen. Ein Inferenz-Cluster verdient dedizierte Geräte, mindestens aber einen Rechner, auf dem sonst nichts Speicherhungriges läuft.

Eine asymmetrische Verteilung ist nur ein Pflaster

Als der Speicher auf dem Master knapp wurde, haben wir versucht, dem Slave einen größeren Anteil des Modells zuzuweisen, etwa im Verhältnis 30 zu 70. Das lindert das Symptom, löst aber das Problem nicht. Die nachhaltige Lösung war, den Master von Fremdlasten zu befreien und das Gesamtbudget realistisch zu planen.

macOS hat eigene Stolpersteine

Drei Dinge, die in keiner MLX-Anleitung stehen: Erstens scheitern manche Systembefehle im Terminal an Berechtigungen, etwa systemsetup ohne Festplattenvollzugriff oder networksetup mit einem Fehler zur Systemkonfigurationsdatenbank. Der Weg über die grafischen Systemeinstellungen war bei uns durchweg zuverlässiger. Zweitens ist ein Gerät am Anmeldebildschirm für den Cluster praktisch offline, automatische Anmeldung und deaktivierter Ruhezustand gehören deshalb zur Grundkonfiguration. Drittens lohnt es sich, alle Netz- und Autostart-Einstellungen sofort neustartfest zu machen, sonst beginnt die Fehlersuche nach jedem Neustart von vorn.

Große Downloads sind ein eigenes Arbeitspaket

Downloads jenseits der 100 GB brechen ab, dauern Stunden und blockieren den Abend. Was geholfen hat: das xet-Backend des Hugging-Face-Clients deaktivieren, Downloads im Hintergrund mit Fortschrittskontrolle laufen lassen, die Geräte mit caffeinate wach halten und das Modell nur einmal herunterladen, um es dann lokal über Thunderbolt zu spiegeln.

Monitoring-Werte kritisch lesen

Die dauerhafte 100-Prozent-GPU-Anzeige im verteilten Leerlauf hat uns anfangs zu einer Fehlersuche verleitet, die keine war. Wer die Eigenheit kennt, spart sich die Sorge und nutzt bessere Indikatoren.

Offline-Betrieb erzeugt Warnungen, keine Fehler

Ohne Internetzugang füllt sich das Log des Agenten mit Meldungen über nicht erreichbare Modellkataloge externer Anbieter. Die lokale Funktion bleibt davon unberührt. Diese Meldungen einmal bewusst einzuordnen verhindert, dass man ihnen später im Störungsfall hinterherläuft.

Oberflächen zeigen nicht automatisch alles

Der frisch eingerichtete Cluster-Endpunkt tauchte in den Modelllisten der angebundenen Werkzeuge nicht von selbst auf, auch ein Aktualisieren-Knopf half nicht. Solche Einträge muss man in der Konfiguration von Hand ergänzen. Das gilt vermutlich für viele Werkzeuge, die primär für Cloud-Modelle gebaut wurden.

Dokumentation entsteht am besten nebenbei

Wir haben Aufbau und Administration weitgehend über einen KI-Coding-Agenten abgewickelt, der jeden Schritt protokolliert hat. Diese Protokolle waren die Grundlage für dieses Manual und mehrfach die Rettung bei der Frage, was vor Wochen eigentlich genau konfiguriert wurde. Wer klassisch von Hand administriert, sollte sich denselben Dienst mit einem konsequent geführten Betriebstagebuch erweisen.

8. Checkliste für den Nachbau

1) Zwei baugleiche Apple-Silicon-Macs bereitstellen, macOS aktualisieren, speicherhungrige Dienste entfernen oder stoppen.
2) Ruhezustand deaktivieren und automatische Anmeldung aktivieren, auf beiden Geräten.
3) Entfernte Anmeldung (SSH) über die Systemeinstellungen aktivieren, SSH-Schlüssel einrichten.
4) Thunderbolt-Kabel verbinden, Thunderbolt-Bridge mit festen IPs einrichten, Einstellung neustartfest machen.
5) RDMA auf beiden Geräten aktivieren und beide Geräte neu starten.

6) Python-Umgebung mit MLX und mlx-lm auf beiden Geräten in identischer Version installieren.
7) Modell nach den Kriterien aus Kapitel 4 auswählen: Speicherbudget, Verteilbarkeit, Tool-Calling.
8) Modell einmal herunterladen und über Thunderbolt auf das zweite Gerät spiegeln.
9) Cluster mit fest begrenztem Kontextfenster starten, API mit Testanfrage prüfen.
10) Agent installieren, auf den lokalen Endpunkt konfigurieren, Autostart einrichten.
11) Telegram-Bot anlegen, Zugriff auf definierte Nutzerkennungen beschränken, Ende-zu-Ende testen.
12) Prüfroutine für Neustarts dokumentieren und einmal komplett durchspielen.

9. Glossar

Begriff	Erklärung
Agent	KI-Anwendung, die ein Sprachmodell mit Werkzeugen verbindet und damit Aufgaben selbstständig ausführen kann, etwa Dateien lesen oder Nachrichten senden.
JACCL	Kommunikations-Backend von MLX für den verteilten Betrieb über mehrere Apple-Geräte hinweg.
Kontextfenster	Die Menge an Text (gemessen in Token), die das Modell gleichzeitig im Blick behalten kann. Bestimmt maßgeblich den Speicherbedarf im Betrieb.
KV-Cache	Zwischenspeicher des Modells für den bisherigen Gesprächsverlauf. Wächst mit der Länge des Kontexts und konkurriert mit den Modellgewichten um den Arbeitsspeicher.
MLX / mlx-lm	Quelloffenes Machine-Learning-Framework von Apple für Apple Silicon sowie das darauf aufbauende Paket zum Betrieb von Sprachmodellen.
MoE (Mixture of Experts)	Modellarchitektur, bei der pro Token nur ein Teil der Parameter aktiv rechnet. Schnell im Betrieb, benötigt im Speicher aber die volle Modellgröße.
Quantisierung	Verfahren zur Verkleinerung eines Modells durch Speicherung der Gewichte mit reduzierter Genauigkeit, üblich sind 8, 4 oder 3 Bit.
RDMA	Remote Direct Memory Access. Direkter Speicherzugriff zwischen zwei Rechnern, unter macOS über Thunderbolt möglich. Senkt die Latenz im Cluster deutlich.
Token	Kleinste Verarbeitungseinheit eines Sprachmodells, im Deutschen grob ein Wortbestandteil. 1.000 Token entsprechen etwa 750 Wörtern.
Tool Calling	Fähigkeit eines Modells, strukturierte Funktionsaufrufe zu erzeugen, mit denen ein Agent Werkzeuge ansteuert. Voraussetzung für agentische Anwendungen.
Unified Memory	Gemeinsamer Speicher von CPU und GPU bei Apple Silicon. Erlaubt es, sehr große Modelle ohne separate Grafikkarte zu betreiben.

10. Über den Demonstrator

KARL ist ein Demonstrator des Mittelstand-Digital Zentrums Fokus Mensch. Er zeigt, dass ein leistungsfähiger KI-Assistent mit Agentenfunktionen vollständig auf eigener Hardware betrieben werden kann, mit voller Datenhoheit und ohne laufende Kosten für externe KI-Dienste. Unternehmen, die den Demonstrator kennenlernen oder den Aufbau vertiefen möchten, finden Ansprechpartner und Termine unter www.digitalzentrum-fokus-mensch.de.