

INSTALLATIONS- UND BETRIEBSANLEITUNG

Lokale KI-Plattform auf dem eigenen Server

Open-Source-Sprachmodelle mit Weboberfläche,
Websuche und Dokumentensuche – vollständig auf Ihrer
eigenen Hardware.

Version 1.0
Oktober 2026

Plattform
Ubuntu 24.04 LTS, Docker

Referenz-Hardware
NVIDIA RTX 4000 SFF Ada, 64 GB
RAM

Inhalt

0	Überblick und Voraussetzungen
1	Betriebssystem installieren
2	Server absichern
3	NVIDIA-Treiber und Docker
4	DNS und KI-Stack
5	Open WebUI einrichten
6	Prüfen, sichern, aktuell halten
A	Fehlerbilder und Lösungen
B	Abnahme-Checkliste

Was Sie am Ende haben

Unter `https://ai.ihre-domain.de` läuft eine Chat-Oberfläche mit Benutzerverwaltung. Die Sprachmodelle laufen auf der Grafikkarte Ihres Servers; Chats und Dokumente verlassen das Haus nicht. Nur für die optionale Websuche gehen Suchanfragen ins Internet.

Von außen sind ausschließlich die Ports 22 (SSH), 80 und 443 erreichbar. Alle übrigen Dienste kommunizieren nur im internen Docker-Netz.

Platzhalter in dieser Anleitung

Platzhalter	Ersetzen durch
<code>ai.ihre-domain.de</code>	Die (Sub-)Domain, unter der die Plattform erreichbar sein soll
<code><SERVER-IP></code>	Öffentliche IPv4-Adresse des Servers

Alle Befehle werden als `root` ausgeführt und kommen daher ohne `sudo` aus.

0 Überblick

Vier Container, ein öffentlicher Eingang. Caddy nimmt HTTPS-Anfragen an und reicht sie an Open WebUI weiter; alles andere ist von außen unsichtbar.

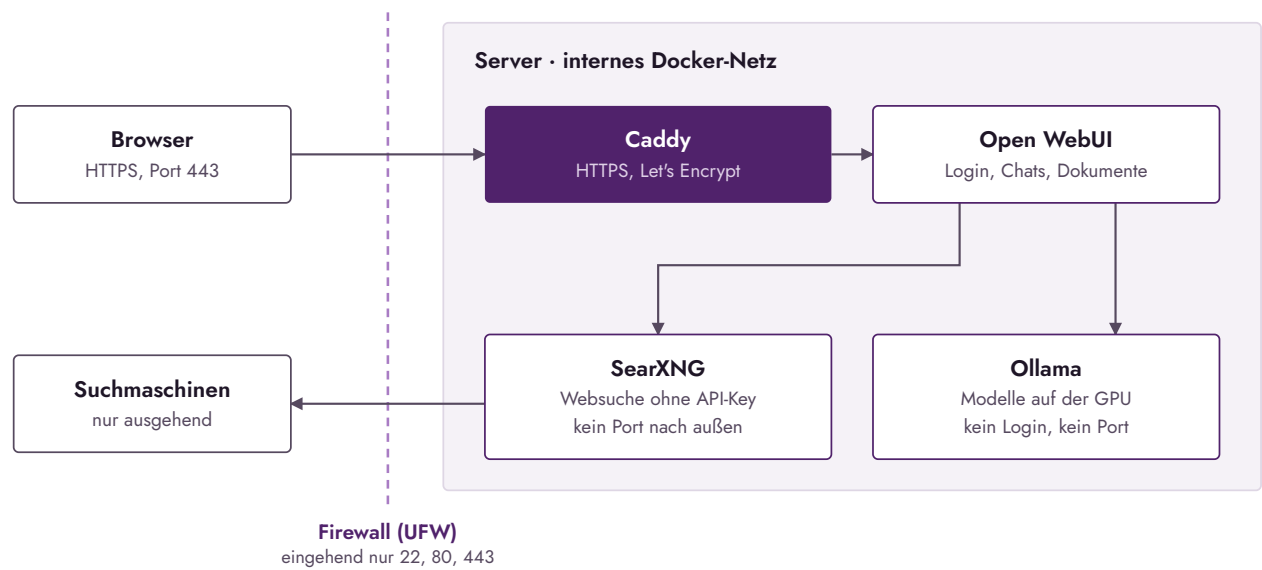


Abbildung 1 · Architektur: Nur Caddy ist aus dem Internet erreichbar.

Sicherheitskonzept

Schicht	Schützt vor
Firewall (UFW, nur 22/80/443)	Zugriff auf alle anderen Ports
SSH nur mit Schlüssel, Fail2ban	Passwort-Angriffe auf den Server-Zugang
Interne Dienste ohne veröffentlichte Ports	Direktzugriff auf die Modell-Schnittstelle (Ollama hat keinen Login)
Caddy mit HTTPS und Security-Headern	Mitlesen von Passwörtern und Chat-Inhalten
Registrierung geschlossen, Freigabe durch Admin	Fremde, die sich selbst ein Konto anlegen

Voraussetzungen

- Dedizierter Server mit NVIDIA-GPU (Referenz: Intel Core i5-13500, 64 GB RAM, 2x NVMe, RTX 4000 SFF Ada mit 20 GB VRAM)
- Ubuntu Server 24.04 LTS
- SSH-Schlüssel auf dem Admin-Rechner
- Zugriff auf die DNS-Einstellungen der Domain

1 Betriebssystem installieren

Installieren Sie **Ubuntu Server 24.04 LTS** (64 bit) auf dem Server und hinterlegen Sie dabei Ihren SSH-Schlüssel für den Benutzer `root`.

Sind zwei Datenträger vorhanden, empfehlen wir ein Software-RAID 1, damit der Server beim Ausfall einer SSD weiterläuft. Anschließend anmelden:

```
ssh root@<SERVER-IP>
```

2 Server absichern

SSH nur per Schlüssel, alle Ports außer 22/80/443 gesperrt. Spätestens vor Schritt 4 erledigen – ab dann ist der Server öffentlich erreichbar.

2.1 System aktualisieren und neu starten

Der Neustart sorgt dafür, dass Schritt 3 bereits auf dem neuesten Kernel läuft.

```
apt update && apt full-upgrade -y
reboot
```

2.2 SSH härten

Root darf sich weiterhin anmelden, aber nur mit Schlüssel.

```
cat > /etc/ssh/sshd_config.d/99-hardening.conf <<'EOF'
PermitRootLogin prohibit-password
PasswordAuthentication no
KbdInteractiveAuthentication no
PubkeyAuthentication yes
MaxAuthTries 3
EOF
sshd -t && systemctl restart ssh
```

Wichtig: Die aktuelle Sitzung offen lassen und in einem zweiten Terminal testen. `ssh root@<SERVER-IP>` muss funktionieren, `ssh -o PubkeyAuthentication=no root@<SERVER-IP>` muss abgelehnt werden. Erst dann die erste Sitzung schließen.

2.3 Host-Firewall (UFW)

```
ufw default deny incoming
ufw default allow outgoing
ufw allow OpenSSH
ufw allow 80/tcp
ufw allow 443/tcp
ufw enable
```

Hinweis: Docker umgeht UFW bei veröffentlichten Ports. Deshalb veröffentlicht in Schritt 4 ausschließlich Caddy Ports; Ollama, SearXNG und Open WebUI bekommen keine.

2.4 Fail2ban gegen Passwort-Angriffe

```
apt install -y fail2ban python3-systemd
cat > /etc/fail2ban/jail.local <<'EOF'
[sshd]
enabled = true
backend = systemd
maxretry = 5
bantime = 1h
EOF
systemctl enable --now fail2ban
fail2ban-client status sshd
```

2.5 Automatische Sicherheitsupdates

```
apt install -y unattended-upgrades
dpkg-reconfigure -plow unattended-upgrades # "Ja" wählen
```

3 NVIDIA-Treiber und Docker

Nach diesem Schritt kann Docker die Grafikkarte nutzen. Installiert wird bewusst nur der Server-Treiber – ohne Desktop-Grafikstack.

Nicht verwenden: `ubuntu-drivers install`. Der Befehl installiert X-Server, `nvidia-prime` und DKMS mit. In unserem Test startete der Server danach nicht mehr und musste neu installiert werden.

3.1 Treiber installieren und ohne Neustart testen

```
# 1. Kernel prüfen und verfügbare Treiber anzeigen
uname -r
ubuntu-drivers list --gpgpu

# 2. Simulieren: die Liste darf kein linux-image, xserver-*, nvidia-prime oder nvidia-dkms
enthalten
apt install -s nvidia-headless-no-dkms-580-server-open nvidia-utils-580-server \
    linux-modules-nvidia-580-server-open-$(uname -r) | grep -E "^Inst" | cut -d' ' -f2

# 3. Installieren
apt install -y nvidia-headless-no-dkms-580-server-open nvidia-utils-580-server \
    linux-modules-nvidia-580-server-open-$(uname -r)

# 4. Ohne Neustart testen: kein neuer Kernel in /boot, Karte mit ca. 20 GB sichtbar
ls /boot
modprobe nvidia && nvidia-smi

# 5. Erst wenn Schritt 4 passt: Neustart und erneut prüfen
reboot
nvidia-smi
```

Die Meldungen `udevadm hwdb is deprecated` bei `ubuntu-drivers` sind harmlos.

3.2 Treiber für Kernel-Updates vorbereiten

Das Meta-Paket installiert bei jedem neuen Kernel automatisch das passende NVIDIA-Modul. Die Simulation darf nur das Paket selbst zeigen.

```
apt install -s linux-modules-nvidia-580-server-open-generic | grep -E "^Inst" | cut -d' ' -f2
apt install -y linux-modules-nvidia-580-server-open-generic
```

3.3 Docker

```
curl -fsSL https://get.docker.com | sh
docker --version
```

3.4 NVIDIA Container Toolkit

```
curl -fsSL https://nvidia.github.io/libnvidia-container/gpgkey \
| gpg --dearmor -o /usr/share/keyrings/nvidia-container-toolkit-keyring.gpg
curl -fsSL https://nvidia.github.io/libnvidia-container/stable/deb/nvidia-container-
toolkit.list \
| sed 's#deb https://#deb [signed-by=/usr/share/keyrings/nvidia-container-toolkit-
keyring.gpg] https://#g' \
> /etc/apt/sources.list.d/nvidia-container-toolkit.list
apt update && apt install -y nvidia-container-toolkit
nvidia-ctk runtime configure --runtime=docker
systemctl restart docker
```

3.5 Test

```
docker run --rm --gpus all ubuntu nvidia-smi # muss dieselbe Tabelle wie in 3.1 zeigen
```

4 DNS und KI-Stack

Vier Dateien in `/opt/ai`, ein Startbefehl. Nur Caddy ist von außen erreichbar.

4.1 DNS-Eintrag

Einen A-Record `ai.ihre-domain.de` → `<SERVER-IP>` anlegen. Erst weitermachen, wenn die Prüfung die richtige IP zeigt – sonst schlägt das Zertifikat fehl, und Let's Encrypt sperrt nach mehreren Fehlversuchen vorübergehend.

```
dig +short ai.ihre-domain.de
```

4.2 Verzeichnis und Schlüssel

```
mkdir -p /opt/ai/searxng && cd /opt/ai
cat > .env <<EOF
DOMAIN=ai.ihre-domain.de
WEBUI_SECRET_KEY=$(openssl rand -hex 32)
SEARXNG_SECRET=$(openssl rand -hex 32)
EOF
chmod 600 .env
```

4.3 Compose-Datei

```
cat > /opt/ai/docker-compose.yml <<'EOF'
services:
  caddy:
    image: caddy:2
    restart: unless-stopped
    ports: ["80:80", "443:443", "443:443/udp"]
    environment: [DOMAIN=${DOMAIN}]
    volumes:
      - ./Caddyfile:/etc/caddy/Caddyfile:ro
      - caddy_data:/data
      - caddy_config:/config
    depends_on: [open-webui]

  open-webui:
    image: ghcr.io/open-webui/open-webui:main
    restart: unless-stopped
    environment:
      - OLLAMA_BASE_URL=http://ollama:11434
      - WEBUI_URL=https://${DOMAIN}
      - WEBUI_SECRET_KEY=${WEBUI_SECRET_KEY}
      - DEFAULT_USER_ROLE=pending
    volumes: [open-webui:/app/backend/data]
    depends_on: [ollama, searxng]

  ollama:
    image: ollama/ollama:latest
    restart: unless-stopped
    environment:
      - OLLAMA_FLASH_ATTENTION=1
      - OLLAMA_KV_CACHE_TYPE=q8_0
      - OLLAMA_CONTEXT_LENGTH=16384
      - OLLAMA_KEEP_ALIVE=30m
    volumes: [ollama:/root/.ollama]
    deploy:
      resources:
        reservations:
          devices:
            - driver: nvidia
              count: all
              capabilities: [gpu]

  searxng:
    image: searxng/searxng:latest
    restart: unless-stopped
    environment: [SEARXNG_SECRET=${SEARXNG_SECRET}]
    volumes: [./searxng:/etc/searxng]

volumes: {caddy_data: {}, caddy_config: {}, open-webui: {}, ollama: {}}
EOF
```

4.4 Caddy und SearXNG konfigurieren

Caddy holt das HTTPS-Zertifikat automatisch. SearXNG liefert JSON für Open WebUI und braucht keinen Rate-Limiter, da es nur intern erreichbar ist.

```

cat > /opt/ai/Caddyfile <<'EOF'
{$DOMAIN} {
    encode zstd gzip
    header {
        Strict-Transport-Security "max-age=31536000; includeSubDomains"
        X-Content-Type-Options "nosniff"
        X-Frame-Options "SAMEORIGIN"
        Referrer-Policy "strict-origin-when-cross-origin"
        -Server
    }
    reverse_proxy open-webui:8080
}
EOF

cat > /opt/ai/searxng/settings.yml <<'EOF'
use_default_settings: true
server:
    limiter: false
    image_proxy: false
search:
    formats: [html, json]
EOF

```

4.5 Starten

```

cd /opt/ai
docker compose up -d
docker compose ps          # 4 Container "Up", nur Caddy mit 0.0.0.0-Ports
docker compose logs -f caddy # auf "certificate obtained successfully" warten, dann Strg+C

```

Sofort weiter mit 5.1: Solange noch kein Admin-Konto existiert, könnte sich jeder mit dem Link als Erster registrieren und wäre damit Administrator.

5 Open WebUI einrichten

Erst das Admin-Konto sichern, dann Registrierung schließen, dann Modelle und Websuche. Alle Einstellungen im Admin-Panel gelten global für alle Benutzer.

5.1 Admin-Konto anlegen

`https://ai.ihre-domain.de` öffnen und sofort ein Konto anlegen. Das erste Konto wird automatisch Administrator.

5.2 Registrierung schließen

Admin-Panel → *Einstellungen* → *Allgemein*: „Neue Registrierungen“ ausschalten, Standardrolle für neue Benutzer auf *pending* lassen. Weitere Benutzer legen Sie unter *Admin-Panel* → *Benutzer* → + selbst an. Diese Werte nur im Admin-Panel ändern – nach dem ersten Start greifen Änderungen an Umgebungsvariablen nicht mehr.

5.3 Modelle laden

Ollama läuft im Container. Ein Alias macht den Befehl auf dem Host aus jedem Verzeichnis nutzbar.

```
echo "alias ollama='docker compose -f /opt/ai/docker-compose.yml exec ollama ollama'" >>
~/.bashrc
source ~/.bashrc

ollama pull gpt-oss:20b          # ca. 14 GB - Hauptmodell
ollama pull qwen3:14b           # ca. 9 GB - Hintergrundaufgaben
ollama pull nomic-embed-text    # Embeddings für Dokumente
ollama run gpt-oss:20b "Sag kurz Hallo"
ollama ps                       # PROCESSOR muss "100% GPU" zeigen
```

`ollama ps` zeigt nur Modelle, die gerade geladen sind. Ohne vorherige Anfrage ist die Liste leer.

5.4 Modelle zuordnen und freigeben

Einstellung	Wert
Admin-Panel → Modelle	<code>gpt-oss:20b</code> als Standard; <code>nomic-embed-text</code> ausschalten (reines Embedding-Modell, gehört nicht in die Chat-Auswahl)
Admin-Panel → Interface → Task-Modell	<code>qwen3:14b</code> (lokal und extern) – erzeugt Chat-Titel und Suchbegriffe
Modell bearbeiten → Sichtbarkeit	<i>Public</i> für <code>gpt-oss:20b</code> und <code>qwen3:14b</code> ; neue Modelle sind sonst nur für den Admin sichtbar
Admin-Panel → Benutzer	Neue Konten von <i>pending</i> auf <i>user</i> setzen

Jeder Benutzer kann im Chat ein anderes Modell wählen und über den Link „Als Standard festlegen“ zu seiner persönlichen Voreinstellung machen. Diese hat Vorrang vor dem Admin-Standard. Der grüne Punkt neben

einem Modell bedeutet nur, dass es gerade im GPU-Speicher geladen ist.

5.5 Websuche

Admin-Panel → *Einstellungen* → *Websuche*: aktivieren, Suchmaschine `searxng`, Query-URL:

```
http://searxng:8080/search?q=<query>
```

Im Chat schaltet das Globus-Symbol die Suche pro Anfrage ein. Für jeden neuen Chat aktiv: *Admin-Panel* → *Modelle* → *Model Defaults* → *Default Features* → *Web Search*.

5.6 Systemprompt mit aktuellem Datum

Die Modelle kennen nur den Stand ihrer Trainingsdaten. Ohne Datum widersprechen sie bei Folgefragen mitunter den eigenen Suchergebnissen. Unter *Modelle* → *Bearbeiten* → *System Prompt* bei beiden Chat-Modellen eintragen:

```
Heute ist {{CURRENT_DATE}}. Dein Trainingswissen endet deutlich früher und ist bei aktuellen Themen wahrscheinlich veraltet. Bei Fragen zu Personen in Ämtern, Ereignissen, Preisen oder allem, was sich seit 2024 geändert haben kann, nutze immer search_web, auch bei Folgefragen. Suchergebnisse und frühere Antworten in diesem Chat, die auf einer Suche beruhen, haben Vorrang vor deinem Trainingswissen. Nenne bei aktuellen Fakten die Quelle.
```

5.7 Dokumente

Admin-Panel → *Einstellungen* → *Dokumente*: Embedding-Engine *Ollama*, Modell `nomie-embed-text`. So laufen auch Embeddings auf der GPU.

Später erweiterbar: Anmeldung über Microsoft Entra ID (OIDC) statt eigener Passwörter sowie kostenpflichtige Cloud-Modelle unter *Admin-Panel* → *Einstellungen* → *Verbindungen* als OpenAI-kompatible API – parallel zu den lokalen Modellen.

6 Prüfen, sichern, aktuell halten

Ein kurzer Test von außen, ein tägliches Backup und ein monatliches Update halten die Plattform sicher und aktuell.

6.1 Von außen prüfen

Auf dem eigenen Rechner ausführen, nicht auf dem Server:

```
nmap -Pn ai.ihre-domain.de # nur 22, 80, 443 "open"
curl -sI https://ai.ihre-domain.de | head # Header mit strict-transport-security
curl -m 5 http://ai.ihre-domain.de:11434 # Ollama: muss ins Timeout laufen
ssh -o PubkeyAuthentication=no root@ai.ihre-domain.de # muss abgelehnt werden
```

6.2 Backup

Wichtig ist nur das Volume `open-webui` mit Benutzern, Chats und Einstellungen. Modelle lassen sich jederzeit neu laden.

```
cat > /opt/ai/backup.sh <<'EOF'
#!/bin/sh
mkdir -p /opt/backup
docker run --rm -v ai_open-webui:/data:ro -v /opt/backup:/backup alpine \
  tar czf /backup/open-webui-$(date +%F).tar.gz -C /data .
find /opt/backup -name 'open-webui-*.tar.gz' -mtime +14 -delete
EOF
chmod +x /opt/ai/backup.sh
(crontab -l 2>/dev/null; echo "30 3 * * * /opt/ai/backup.sh") | crontab -
/opt/ai/backup.sh && ls -lh /opt/backup # einmal manuell testen
```

Empfehlung: Das Backup liegt auf demselben Server und schützt nur vor Fehlbedienung. Für echte Ausfallsicherheit die Archive zusätzlich extern ablegen, z. B. auf einem externen Speicher oder NAS.

6.3 Updates

Das Betriebssystem aktualisiert sich selbst (2.5), das NVIDIA-Modul zieht bei neuen Kerneln mit (3.2). Container etwa monatlich:

```
cd /opt/ai && docker compose pull && docker compose up -d && docker image prune -f
```

Nach Kernel-Updates gelegentlich `reboot` und danach `nvidia-smi` prüfen.

A Fehlerbilder und Lösungen

Diese Probleme sind bei der Referenzinstallation tatsächlich aufgetreten.

Symptom	Ursache und Lösung
Server startet nach Treiberinstallation nicht mehr	<code>ubuntu-drivers install</code> hat den Desktop-Stack installiert. Neu installieren (Schritt 1) und den Treiber nach 3.1 einrichten.
<code>docker compose up</code> : Timeout bei <code>ghcr.io</code>	Meist ein kurzer Aussetzer der GitHub-Registry: <code>docker compose pull open-webui</code> wiederholen. Bleibt es, mit <code>curl -4 -m 10 https://ghcr.io/v2/</code> prüfen (Antwort <code>401</code> = Verbindung in Ordnung) und eine vorgeschaltete Firewall auf blockierte Antwortpakete kontrollieren.
<code>ollama: command not found</code>	Ollama läuft nur im Container. Alias aus 5.3 anlegen oder <code>docker compose exec ollama ollama ...</code> verwenden.
<code>ollama ps</code> ist leer	Kein Fehler – es ist gerade kein Modell geladen. Einmal <code>ollama run</code> oder eine Chat-Anfrage, dann erneut prüfen.
„nomic-embed-text does not support chat“	Das Embedding-Modell ist als Chat- oder Task-Modell ausgewählt. Einstellungen nach 5.4 setzen.
Andere Benutzer sehen keine Modelle	Modelle sind standardmäßig privat. Sichtbarkeit auf <i>Public</i> setzen (5.4).
Neue Benutzer hängen im Wartebildschirm	Rolle <i>pending</i> . Unter <i>Admin-Panel</i> → <i>Benutzer</i> auf <i>user</i> setzen.
Antworten mit veraltetem Wissen trotz Websuche	Prüfen, ob über der Antwort „Explored search_web“ steht. Wenn nein: Modell hat nicht gesucht – Systemprompt aus 5.6 setzen und <code>gpt-oss:20b</code> verwenden, das Suchwerkzeuge zuverlässiger aufruft.
Websuche liefert nichts	SearXNG direkt testen: <code>docker compose exec open-webui curl -s "http://searxng:8080/search?q=test&format=json" head -c 500</code> Bei leeren <code>results</code> weitere Engines (z. B. DuckDuckGo, Brave) in <code>settings.yml</code> aktivieren.
<code>systemctl enable nvidia-persistenced</code> meldet „no installation config“	Kein Fehler. Der Dienst startet automatisch; der Persistence-Modus ist für diesen Einsatz nicht nötig.

B Abnahme-Checkliste

Alle Punkte abhaken, bevor die Plattform an die Benutzer übergeben wird.

- | | |
|---|---|
| ☐ Ubuntu 24.04 LTS installiert, aktuell | ☐ Admin-Konto angelegt, Registrierung geschlossen |
| ☐ SSH-Login ohne Schlüssel abgelehnt | |
| ☐ UFW aktiv, Fail2ban läuft | ☐ <code>ollama ps</code> zeigt 100 % GPU |
| ☐ <code>nvidia-smi</code> funktioniert nach Neustart | ☐ Embedding-Modell ausgeblendet, Task-Modell gesetzt |
| ☐ NVIDIA-Meta-Paket installiert | ☐ Chat-Modelle auf <i>Public</i> |
| ☐ nmap zeigt nur 22, 80, 443 | ☐ Systemprompt mit Datum gesetzt |
| ☐ HTTPS-Zertifikat gültig | ☐ Websuche liefert Quellen |
| | ☐ Backup-Cronjob läuft, erstes Archiv vorhanden |

Moregate Software

Tools für Teams, die ihre Daten lieber im eigenen Haus behalten.

moregate-software.com