Open WebUI Not Connecting to Ollama? Every Fix

February 26, 2026 · by Mark Bartlett

Download this guide as PDF

Quick Answer: The #1 cause: Open WebUI runs in Docker and `localhost` inside the container means the container, not your machine. Fix for Mac/Windows: set `OLLAMA_BASE_URL=http://host.docker.internal:11434`. Fix for Linux: run with `--network=host`. If you installed Open WebUI with pip and Ollama is on the same machine, it usually just works — check that Ollama is actually running with `curl http://localhost:11434`.

📚 **More on this topic:** Open WebUI Setup Guide · Ollama Troubleshooting · Ollama API Connection Refused · VRAM Requirements

You installed Open WebUI. You installed Ollama. Ollama works fine in the terminal. But Open WebUI shows “Could not connect to Ollama” or just a blank model list.

I’ve seen this question more than any other Open WebUI issue. It’s almost always a networking problem between the two, and the fix is usually one environment variable or one Docker flag. But there are about eight variations depending on how you installed things and what OS you’re on.

Every version, with the exact fix.

Quick Diagnostic

Figure out which layer is broken first:

```
# Step 1: Is Ollama running at all?
curl http://localhost:11434
# Should print "Ollama is running"

# Step 2: Does Ollama have models?
curl http://localhost:11434/api/tags
# Should print JSON with your model list

# Step 3: Can Open WebUI's container reach Ollama?
docker exec open-webui curl http://host.docker.internal:11434
```

```
# Or if using --network=host:
docker exec open-webui curl http://127.0.0.1:11434
```

Step 1 fails? Ollama isn't running. Start it. Step 2 fails? Ollama is running but has no models. Pull one with `ollama pull qwen3:8b`. Step 3 fails? That's the networking problem. Read on.

If you installed Open WebUI with pip (not Docker), skip to pip install issues.

The #1 Problem: "Could Not Connect to Ollama"

This error means Open WebUI tried to reach Ollama at whatever URL it has configured and got nothing back.

90% of the time it's this: **Open WebUI runs in a Docker container, and `localhost` inside that container means the container itself, not your host machine.** Ollama is running on your host. The container can't see it.

Fix for Mac and Windows (Docker Desktop)

Use `host.docker.internal`, Docker Desktop's special hostname that resolves to the host machine:

```
docker run -d -p 3000:8080 \
  --add-host=host.docker.internal:host-gateway \
  -e OLLAMA_BASE_URL=http://host.docker.internal:11434 \
  -v open-webui:/app/backend/data \
  -e WEBUI_SECRET_KEY=your-secret-key \
  --name open-webui --restart always \
  ghcr.io/open-webui/open-webui:main
```

The key parts:

- `--add-host=host.docker.internal:host-gateway` makes the hostname available
- `OLLAMA_BASE_URL=http://host.docker.internal:11434` tells Open WebUI where to find Ollama

Fix for Linux

`host.docker.internal` works on Docker Engine 20.10+ on Linux, but the more reliable approach is host networking:

```
docker run -d --network=host \
  -e OLLAMA_BASE_URL=http://127.0.0.1:11434 \
  -v open-webui:/app/backend/data \
  -e WEBUI_SECRET_KEY=your-secret-key \
  --name open-webui --restart always \
  ghcr.io/open-webui/open-webui:main
```

With `--network=host`, the container shares the host's network stack. `localhost` and `127.0.0.1` point where you'd expect. The tradeoff: the UI listens on port **8080** instead of 3000 (because `-p 3000:8080` port mapping doesn't apply with host networking).

Fix for pip Install

If you installed Open WebUI with `pip install open-webui` or `uvx`, there's no Docker container. Open WebUI runs directly on your machine. `localhost` means your machine. So if Ollama is running on the same box, it should just work.

If it doesn't:

```
# Make sure Ollama is actually running
curl http://localhost:11434

# If that works, check Open WebUI's settings
# Go to Admin → Settings → Connections → Ollama
# Verify the URL is http://localhost:11434
```

Common pip gotcha: you started Ollama with a custom `OLLAMA_HOST` on a different port. Match the port in Open WebUI's settings.

Ollama Running but No Models Show Up

Open WebUI connects (no error message) but the model dropdown is empty or stuck loading.

Cause 1: Ollama Has No Models

Obvious but worth checking:

```
ollama list
```

If that's empty, pull a model:

```
ollama pull qwen3:8b
```

Then click the refresh icon next to the model dropdown in Open WebUI.

Cause 2: Ollama Only Listening on 127.0.0.1

Ollama binds to `127.0.0.1` by default. If Open WebUI is reaching it from a different network interface (Docker bridge network, different machine, WSL2), the connection gets refused even though Ollama is running.

Fix: Tell Ollama to listen on all interfaces:

```
# Linux (systemd)
sudo systemctl edit ollama.service
```

Add under `[Service]`:

```
[Service]
Environment="OLLAMA_HOST=0.0.0.0:11434"
```

Then:

```
sudo systemctl daemon-reload
sudo systemctl restart ollama
```

On macOS:

```
launchctl setenv OLLAMA_HOST "0.0.0.0:11434"
# Restart Ollama from the menu bar
```

On Windows: set `OLLAMA_HOST` to `0.0.0.0:11434` in System Environment Variables, then restart Ollama from the system tray.

Cause 3: CORS Blocking the Request

If Open WebUI connects through the OpenAI-compatible API endpoint or a non-standard setup, CORS headers can block the model list request.

Fix:

```
# Allow Open WebUI's origin
OLLAMA_ORIGINS=* ollama serve
```

Or set it permanently the same way as `OLLAMA_HOST` above. The wildcard `*` allows all origins. Fine for local use. Don't do this on a public-facing server.

Cause 4: Open WebUI Cached a Bad Config

Open WebUI stores connection settings in its database. If you changed the Ollama URL via environment variable but previously saved a different URL through the admin panel, the database value wins.

Fix — nuclear option:

```
docker run -d \
  -e RESET_CONFIG_ON_START=true \
  -e OLLAMA_BASE_URL=http://host.docker.internal:11434 \
  ...
```

Or set `ENABLE_PERSISTENT_CONFIG=false` to always use environment variables instead of database-stored settings.

Cause 5: Model List Timeout

Open WebUI waits up to 10 seconds for the model list endpoint by default. If Ollama is slow to respond (cold start, overloaded machine), the request times out silently and shows an empty list.

Fix: Lower the timeout so it fails fast and retries, or increase it if your Ollama server is genuinely slow:

```
-e AIOHTTP_CLIENT_TIMEOUT_MODEL_LIST=3    # Fail fast (3 seconds)
# or
-e AIOHTTP_CLIENT_TIMEOUT_MODEL_LIST=30   # Wait longer
```

Docker-Specific Connection Problems

Wrong OLLAMA_BASE_URL

This is the variable that tells Open WebUI where Ollama lives. Get it wrong and nothing works.

Your Setup	Correct OLLAMA_BASE_URL
Ollama on host, Open WebUI in Docker (Mac/Win)	`http://host.docker.internal:11434`
Ollama on host, Open WebUI in Docker (Linux)	`http://127.0.0.1:11434` (with `--network=host`)
Both in Docker Compose, same network	`http://ollama:11434` (use the service name)
Ollama on a different machine	`http://<ollama-ip>:11434`

Common mistakes:

- `http://localhost:11434` — points to the container, not the host
- `http://ollama:11434` — only works if Ollama is a Docker service on the same compose network
- Missing the `http://` prefix
- Adding a trailing slash — some versions choke on `http://host.docker.internal:11434/`

Bridge vs Host vs Custom Network

Docker has three networking modes. Each changes how Open WebUI talks to Ollama.

Mode	Docker Flag	How localhost Works	Best For
Bridge (default)	None or `--network=bridge`	Points to container. Use `host.docker.internal`	Mac/Windows
Host	`--network=host`	Points to host machine	Linux
Custom	`--network=my-network`	Use container names as hostnames	Docker Compose

If you're on Linux and nothing else works, `--network=host` is the simplest fix. It bypasses all Docker network isolation.

The Docker Compose Setup That Works

If you want Ollama and Open WebUI in containers on the same Compose network:

```
services:
  ollama:
    image: ollama/ollama:latest
    restart: unless-stopped
    volumes:
      - ollama:/root/.ollama
    # GPU passthrough for NVIDIA:
    # deploy:
    #   resources:
    #     reservations:
    #       devices:
    #         - capabilities: [gpu]

  open-webui:
    image: ghcr.io/open-webui/open-webui:main
    ports:
      - "3000:8080"
    environment:
      - OLLAMA_BASE_URL=http://ollama:11434
      - WEBUI_SECRET_KEY=your-secret-key
    depends_on:
      - ollama
    volumes:
      - open-webui:/app/backend/data
    restart: unless-stopped
```

```
volumes:
  ollama: {}
  open-webui: {}
```

The key: `OLLAMA_BASE_URL=http://ollama:11434` uses the service name `ollama` as the hostname. Docker's internal DNS resolves it to the Ollama container's IP. No `host.docker.internal` needed because they're on the same Docker network.

Older Docker on Linux

Docker Engine versions before 20.10 don't support `host.docker.internal` on Linux. If you're on an older version and can't use `--network=host`, you have two options:

1. **Get the host IP manually:**

```
# From inside the container
ip route | grep default | awk '{print $3}'
# Use that IP as OLLAMA_BASE_URL
```

1. **Upgrade Docker.** Seriously. Docker 20.10 came out in 2020.

WSL2 + Docker Desktop

This one is its own special headache. WSL2 and Docker Desktop each run in separate network namespaces on Windows. Ollama running in one can't always reach things in the other.

Ollama in WSL2, Open WebUI in Docker Desktop

The most common scenario. You installed Ollama natively in WSL2 (maybe Ubuntu), and Open WebUI in Docker Desktop.

Problem: Docker Desktop on Windows runs its own Linux VM. It can't see WSL2's `localhost`.

Fix — Option A: Run both in WSL2. Install Docker Engine directly in WSL2 (not Docker Desktop) and run Open WebUI there. Then `localhost` works everywhere.

Fix — Option B: Use the Windows host IP to bridge them:

```
# In WSL2, get your Windows host IP:
WIN_IP=$(cat /etc/resolv.conf | grep nameserver | awk '{print $2}')
echo $WIN_IP

# Make Ollama listen on all interfaces (in WSL2):
OLLAMA_HOST=0.0.0.0:11434 ollama serve

# In Docker Desktop, set:
OLLAMA_BASE_URL=http://<that-windows-ip>:11434
```

Fix — Option C: Run Open WebUI in WSL2 with pip instead of Docker:

```
# In WSL2
pip install open-webui
open-webui serve
# localhost:8080 in your browser, connects to Ollama on localhost:11434
```

No Docker networking to debug at all.

Ollama on Windows Native, Open WebUI in Docker

If you're running the Ollama Windows installer (not WSL2), and Open WebUI in Docker Desktop:

```
docker run -d -p 3000:8080 \
  -e OLLAMA_BASE_URL=http://host.docker.internal:11434 \
  -v open-webui:/app/backend/data \
  --name open-webui \
  ghcr.io/open-webui/open-webui:main
```

This usually works because Docker Desktop's `host.docker.internal` resolves to the Windows host where Ollama is listening. If it doesn't, make sure Ollama is set to `OLLAMA_HOST=0.0.0.0:11434` in Windows environment variables.

Remote Ollama Server

Running Ollama on one machine (maybe a GPU server) and Open WebUI on another (a laptop, a Pi, whatever).

Setup

On the Ollama machine:

1. Set Ollama to listen on all interfaces:

```
# Linux
sudo systemctl edit ollama.service
# Add: Environment="OLLAMA_HOST=0.0.0.0:11434"
sudo systemctl daemon-reload && sudo systemctl restart ollama
```

1. Open the firewall:

```
sudo ufw allow 11434/tcp
```

1. Verify from the remote machine:

```
curl http://<ollama-machine-ip>:11434/api/tags
```

On the Open WebUI machine:

```
docker run -d -p 3000:8080 \
  -e OLLAMA_BASE_URL=http://<ollama-machine-ip>:11434 \
  -v open-webui:/app/backend/data \
  --name open-webui --restart always \
  ghcr.io/open-webui/open-webui:main
```

Or set the URL in Admin → Settings → Connections → Ollama after first launch.

Security Warning

Ollama has no authentication. If you set `OLLAMA_HOST=0.0.0.0`, anyone who can reach port 11434 can use your GPU to run models. On a home network that's probably fine. On a VPS or office network, don't expose it directly.

Options:

- **Firewall rules** to restrict which IPs can connect
- **SSH tunnel:** `ssh -L 11434:localhost:11434 ollama-server` — access it as if it were local
- **Reverse proxy** (Nginx/Caddy) with basic auth in front of Ollama

Don't expose Ollama to the public internet without one of these.

"Failed to Fetch" Errors Mid-Conversation

You're chatting, everything works, then a response fails with "Failed to fetch" or "Network error."

Cause 1: Ollama Timeout on Long Generations

Large models generating long responses take time. If the connection between Open WebUI and Ollama times out before the response finishes, you get a fetch error.

Fix: Increase Ollama's keep-alive and request timeout:

```
OLLAMA_KEEP_ALIVE=-1 ollama serve
```

`-1` means models stay loaded indefinitely. The default is 5 minutes — if Ollama idles for 5 minutes between tokens (shouldn't happen, but edge cases exist), it unloads the model mid-response.

Cause 2: Model Crashed from OOM

The model ran out of memory mid-generation. This happens when context fills up and the KV cache pushes VRAM usage past what's available.

Diagnose:

```
# Check Ollama logs
journalctl -u ollama --no-pager -n 50     # Linux systemd
docker logs ollama --tail 50               # Docker

# Look for "out of memory" or "cudaMalloc failed"
```

Fix: Reduce context length or use a smaller model. See our VRAM requirements guide for what fits in your GPU.

Cause 3: Docker Desktop Resource Limits

Docker Desktop (Mac/Windows) defaults to limited memory. If the Open WebUI container runs out of its allocated RAM, requests fail.

Fix: Docker Desktop → Settings → Resources → increase memory to at least 4 GB. More if you're using Open WebUI's built-in features like local Whisper or RAG embeddings.

Connection Works Then Randomly Drops

Everything is fine for a while, then models disappear from the dropdown or responses start failing.

Cause 1: Ollama Unloaded the Model

Ollama's default `OLLAMA_KEEP_ALIVE` is 5 minutes. After 5 minutes of no requests, the model unloads from VRAM. The next request takes 10-30 seconds to reload, which can look like a connection failure.

Fix:

```
# Keep models loaded forever
OLLAMA_KEEP_ALIVE=-1

# Or set a longer timeout
OLLAMA_KEEP_ALIVE=60m
```

Set this as an environment variable permanently (systemd edit, launchctl, Windows env vars, or Docker `-e` flag).

Cause 2: Docker Container Restarting

Check if the Open WebUI container is crash-looping:

```
docker ps -a | grep open-webui
# Look at the STATUS column — "Restarting" is bad

docker logs open-webui --tail 50
# Check for errors
```

Common causes: out of memory, database corruption, port conflicts. Usually means recreating the container with more resources or a fresh volume.

Cause 3: Ollama Service Crashed

On Linux, the systemd service might have restarted silently:

```
sudo systemctl status ollama
journalctl -u ollama --no-pager -n 20
```

If it's crashing repeatedly, you likely have a driver issue or OOM situation. Check our Ollama troubleshooting guide for GPU and memory diagnostics.

Slow Responses (Not a Connection Problem)

If Open WebUI connects fine and models load, but responses crawl, that's not a connection issue. It's a performance problem on the Ollama side.

Quick check:

```
ollama ps
```

Look at the **Processor** column:

What You See	What It Means
`100% GPU`	Full GPU. Normal speed.
`100% CPU`	Running on CPU. Expect 2-8 tok/s instead of 40+.
`48%/52% CPU/GPU`	Partially offloaded. Faster than CPU, but 3-5x slower than full GPU.

If you see CPU or a split, the model doesn't fully fit in your VRAM. Options: reduce context length, use a smaller quantization (Q4 instead of Q6), use a smaller model, or upgrade your GPU.

For the full performance troubleshooting flow, see our Ollama troubleshooting guide or Ollama not using GPU fix.

Quick Reference Table

Problem	Most Likely Cause	Fix
"Could not connect to Ollama"	Docker localhost confusion	Use `host.docker.internal` or `--network=host`
Connection refused	Ollama isn't running	`ollama serve` or `systemctl start ollama`
Connected but no models	Ollama on 127.0.0.1 only	Set `OLLAMA_HOST=0.0.0.0`
No models (Docker Compose)	Wrong service URL	Use `http://ollama:11434`
WSL2 can't connect	Network isolation	Use Windows host IP or run both in WSL2
Remote server won't connect	Firewall or binding	`OLLAMA_HOST=0.0.0.0` + open port 11434
"Failed to fetch" mid-chat	OOM or timeout	Check logs, reduce context, increase keep-alive
Random disconnects	Model unloaded	`OLLAMA_KEEP_ALIVE=-1`
Slow but connected	Running on CPU	Check `ollama ps`, see VRAM guide
Config changes don't stick	DB overrides env vars	Set `RESET_CONFIG_ON_START=true`

The Bottom Line

Almost every Open WebUI + Ollama connection problem comes down to one thing: the URL that Open WebUI uses to reach Ollama is wrong for your network setup.

The debug flowchart:

1. **Is Ollama running?** `curl http://localhost:11434` on the Ollama machine
2. **Can Open WebUI reach it?** `docker exec open-webui curl http://<your-ollama-url>:11434`
3. **No?** Fix the URL: `host.docker.internal` for Mac/Windows Docker, `127.0.0.1` with `--network=host` for Linux, service name for Docker Compose, actual IP for remote servers
4. **Still no?** Set `OLLAMA_HOST=0.0.0.0` on the Ollama side and check your firewall

Once the URL is right, it just works. Every time you change your setup (new machine, different Docker config, WSL2), you'll re-check the URL. But the logic stays the same: figure out what IP Open WebUI needs to reach Ollama, and point `OLLAMA_BASE_URL` at it.

For the full Open WebUI setup walkthrough, see our setup guide. For Ollama problems beyond connectivity, see the Ollama troubleshooting guide.

Get notified when we publish new guides.

Subscribe — free, no spam

Source: https://insiderllm.com/guides/open-webui-ollama-connection-fix/

Free guides for running AI locally