Ollama API Connection Refused: Quick Fixes

February 18, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** First: is Ollama running? Run `ollama serve` or `systemctl start ollama`. If it's running but you can't connect remotely, Ollama binds to localhost by default — set OLLAMA_HOST=0.0.0.0:11434 to open it. Docker containers need host.docker.internal or --network host to reach Ollama on the host. If curl works but your code doesn't, you're probably missing the /api/ prefix in your URL.

📚 **More on this topic:** Ollama Troubleshooting Guide · Open WebUI Setup Guide · Run Your First Local LLM

You're trying to hit Ollama's API — from code, from Docker, from another machine — and getting `connection refused`. The model is loaded. The code looks right. But something between you and port 11434 is broken.

Here's every reason why, in order of likelihood.

Quick Diagnostic

Run this first:

```
curl http://localhost:11434/api/tags && echo "Ollama OK" || echo "Ollama NOT responding"
```

If that prints `Ollama OK` with a JSON list of models, Ollama is running and listening locally. Your problem is networking — skip to section 3.

If it prints `Ollama NOT responding`, start at section 1.

---

1. Ollama Isn't Running

Error:

```
curl: (7) Failed to connect to localhost port 11434: Connection refused
```

Fix:

```
# Linux (systemd)
sudo systemctl start ollama

# Or run manually
ollama serve
```

On macOS, launch the Ollama app from Applications. On Windows, start Ollama from the system tray.

Verify: `curl http://localhost:11434` should return `Ollama is running`.

---

2. Wrong Port

Error: Connection refused, but Ollama is running.

Cause: You're hitting the wrong port. Ollama's default is **11434**, not 11435, not 8080, not 3000.

Fix: Check what port Ollama is actually using:

```
# Linux/macOS
ss -tlnp | grep ollama

# Or check the process
ps aux | grep ollama
```

If you set a custom port via `OLLAMA_HOST`, match it in your client:

```
# If Ollama was started with:
OLLAMA_HOST=0.0.0.0:8080 ollama serve

# Then connect to:
curl http://localhost:8080/api/tags
```

3. Listening on Localhost Only

Error: `curl localhost:11434` works on the Ollama machine, but connecting from another machine (or from a Docker container) gets connection refused.

Cause: By default, Ollama binds to `127.0.0.1:11434` — localhost only. It won't accept connections from other IPs.

Fix:

```
# Linux: edit the systemd service
sudo systemctl edit ollama
```

Add under `[Service]`:

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

Then restart:

```
sudo systemctl restart ollama
```

On macOS, set the environment variable before launching:

```
launchctl setenv OLLAMA_HOST "0.0.0.0:11434"
```

Then restart Ollama from the menu bar.

Verify from another machine:

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

4. Firewall Blocking

Error: `OLLAMA_HOST=0.0.0.0` is set, Ollama is listening on all interfaces, but remote connections still fail.

Cause: Firewall is blocking port 11434.

Fix:

```
# Ubuntu/Debian (ufw)
sudo ufw allow 11434/tcp

# CentOS/RHEL (firewalld)
sudo firewall-cmd --add-port=11434/tcp --permanent
sudo firewall-cmd --reload

# Check with iptables
sudo iptables -L -n | grep 11434
```

On Windows, add an inbound rule in Windows Defender Firewall for TCP port 11434.

5. Docker Container Can't Reach Host Ollama

Error: Your app runs in Docker. Ollama runs on the host. `curl http://localhost:11434` inside the container fails.

Cause: `localhost` inside a Docker container means the container itself, not the host machine.

Fix — Option A (recommended):

```
# Use Docker's special hostname for the host machine
curl http://host.docker.internal:11434/api/tags
```

In docker-compose:

```
environment:
  - OLLAMA_BASE_URL=http://host.docker.internal:11434
```

Fix — Option B: Run the container with host networking:

```
docker run --network host your-app
```

This makes the container share the host's network. `localhost` inside the container means the actual host. Simpler, but less isolated.

Note: `host.docker.internal` works on Docker Desktop (Mac/Windows) and Docker Engine 20.10+ on Linux. On older Linux Docker, use `--network host` or the host's actual IP.

---

6. WSL2 to Windows (or Vice Versa)

Error: Ollama runs on Windows, your app runs in WSL2 (or vice versa). Connection refused.

Cause: WSL2 runs in a lightweight VM with its own network stack. `localhost` doesn't cross the boundary cleanly.

Fix — Ollama on Windows, app in WSL2:

```
# Get the Windows host IP from inside WSL2
WIN_IP=$(cat /etc/resolv.conf | grep nameserver | awk '{print $2}')
curl http://$WIN_IP:11434/api/tags
```

Fix — Ollama in WSL2, app on Windows:

Make sure Ollama binds to all interfaces (`OLLAMA_HOST=0.0.0.0:11434`), then connect from Windows using the WSL2 IP:

```
# Get WSL2 IP
wsl hostname -I
```

7. Open WebUI Can't Connect

Error: Open WebUI shows "Ollama connection error" or models don't load.

Cause: Wrong `OLLAMA_BASE_URL` in the Docker container. This is the most common Open WebUI setup issue.

Fix: In your docker run or docker-compose:

```
# If Ollama is on the host machine
docker run -d \
  -e OLLAMA_BASE_URL=http://host.docker.internal:11434 \
  -p 3000:8080 \
  ghcr.io/open-webui/open-webui:main
```

Common mistakes:

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

See our Open WebUI setup guide for the full docker-compose configuration.

8. API Works in curl but Not in Code

Error: `curl http://localhost:11434/api/generate` works fine. But your Python/JavaScript/Go code gets connection refused or 404.

Cause: Usually a URL formatting issue.

Common mistakes:

```
# Wrong — missing /api/ prefix
requests.post("http://localhost:11434/generate", ...)

# Right
requests.post("http://localhost:11434/api/generate", ...)

# Wrong — using /v1/ (that's OpenAI format, needs /v1/ prefix AND different endpoint)
requests.post("http://localhost:11434/v1/generate", ...)

# Right — OpenAI-compatible endpoint
requests.post("http://localhost:11434/v1/chat/completions", ...)
```

Also check:

- **Content-Type header:** Must be `application/json` for POST requests
- **Request body:** Must be valid JSON, not form-encoded
- **Streaming:** Ollama streams by default. Set `"stream": false` in the request body if you want the complete response at once

---

9. Port Already in Use

Error:

```
listen tcp 127.0.0.1:11434: bind: address already in use
```

Cause: Another Ollama process (or something else) is already using port 11434.

Fix:

```
# Find what's using the port
sudo lsof -i :11434
# or
sudo ss -tlnp | grep 11434

# Kill the old process
sudo kill $(sudo lsof -t -i :11434)
```

```
# Restart Ollama
ollama serve
```

On systemd, leftover manual `ollama serve` processes conflict with the service:

```
# Kill manual process, use the service instead
pkill ollama
sudo systemctl start ollama
```

Quick Reference

Symptom	Fix
Connection refused locally	`ollama serve` or `systemctl start ollama`
Works locally, not remotely	`OLLAMA_HOST=0.0.0.0:11434`
Docker can't connect	Use `host.docker.internal:11434`
WSL2 can't connect	Use Windows host IP from `/etc/resolv.conf`
Open WebUI fails	Set `OLLAMA_BASE_URL=http://host.docker.internal:11434`
curl works, code doesn't	Check URL path — needs `/api/` prefix
Port conflict	`lsof -i :11434`, kill old process

Bottom Line

Connection refused means something between your client and Ollama's listener is broken. In order of likelihood: Ollama isn't running, it's bound to localhost, or Docker networking is doing what Docker networking does. The fix is almost always one environment variable or one flag change.

If none of these fix your issue, check the full Ollama troubleshooting guide — it covers GPU detection, model loading, and performance problems beyond API connectivity.

Source: https://insiderllm.com/guides/ollama-api-connection-refused-fix/

Free guides for running AI locally