Troubleshooting

- LLM libraries
- Installing older or pre-release versions on Linux
- Linux tmp noexec
- Linux docker
- NVIDIA GPU Discovery
  - Linux NVIDIA Troubleshooting
- AMD GPU Discovery
  - AMD Driver Version Mismatch
- Multiple AMD GPUs
- Windows Terminal Errors

Sometimes Ollama may not perform as expected. One of the best ways to figure out what happened is to take a look at the logs. Find the logs on **Mac** by running the command:

```
cat ~/.ollama/logs/server.log
```

On **Linux** systems with systemd, the logs can be found with this command:

```
journalctl -u ollama --no-pager --follow --pager-end
```

When you run Ollama in a **container**, the logs go to stdout/stderr in the container:

```
docker logs <container-name>
```

(Use `docker ps` to find the container name)

If manually running `ollama serve` in a terminal, the logs will be on that terminal.

When you run Ollama on **Windows**, there are a few different locations. You can view them in the explorer window by hitting `<cmd>+R` and type in:

- `explorer %LOCALAPPDATA%\Ollama` to view logs. The most recent server logs will be in `server.log` and older logs will be in `server-#.log`

- `explorer %LOCALAPPDATA%\Programs\Ollama` to browse the binaries (The installer adds this to your user PATH)
- `explorer %HOMEPATH%\.ollama` to browse where models and configuration is stored
- `explorer %TEMP%` where temporary executable files are stored in one or more `ollama*` directories

To enable additional debug logging to help troubleshoot problems, first **Quit the running app from the tray menu** then in a powershell terminal

```
$env:OLLAMA_DEBUG="1"
& "ollama app.exe"
```

Join the Discord for help interpreting the logs.

LLM libraries

Ollama includes multiple LLM libraries compiled for different GPUs and CPU vector features. Ollama tries to pick the best one based on the capabilities of your system. If this autodetection has problems, or you run into other problems (e.g. crashes in your GPU) you can workaround this by forcing a specific LLM library. `cpu_avx2` will perform the best, followed by `cpu_avx` an the slowest but most compatible is `cpu`. Rosetta emulation under MacOS will work with the `cpu` library.

In the server log, you will see a message that looks something like this (varies from release to release):

```
Dynamic LLM libraries [rocm_v6 cpu cpu_avx cpu_avx2 cuda_v11 rocm_v5]
```

Experimental LLM Library Override

You can set OLLAMA_LLM_LIBRARY to any of the available LLM libraries to bypass autodetection, so for example, if you have a CUDA card, but want to force the CPU LLM library with AVX2 vector support, use:

```
OLLAMA_LLM_LIBRARY="cpu_avx2" ollama serve
```

You can see what features your CPU has with the following.

```
cat /proc/cpuinfo| grep flags | head -1
```

Installing older or pre-release versions on Linux

If you run into problems on Linux and want to install an older version, or you'd like to try out a pre-release before it's officially released, you can tell the install script which version to install.

```
curl -fsSL https://ollama.com/install.sh | OLLAMA_VERSION=0.5.7 sh
```

Linux tmp noexec

If your system is configured with the "noexec" flag where Ollama stores its temporary executable files, you can specify an alternate location by setting OLLAMA_TMPDIR to a location writable by the user ollama runs as. For example OLLAMA_TMPDIR=/usr/share/ollama/

Linux docker

If Ollama initially works on the GPU in a docker container, but then switches to running on CPU after some period of time with errors in the server log reporting GPU discovery failures, this can be resolved by disabling systemd cgroup management in Docker. Edit `/etc/docker/daemon.json` on the host and add `"exec-opts": ["native.cgroupdriver=cgroupfs"]` to the docker configuration.

NVIDIA GPU Discovery

When Ollama starts up, it takes inventory of the GPUs present in the system to determine compatibility and how much VRAM is available. Sometimes this discovery can fail to find your GPUs. In general, running the latest driver will yield the best results.

Linux NVIDIA Troubleshooting

If you are using a container to run Ollama, make sure you've set up the container runtime first as described in docker

Sometimes the Ollama can have difficulties initializing the GPU. When you check the server logs, this can show up as various error codes, such as "3" (not initialized), "46" (device unavailable),

"100" (no device), "999" (unknown), or others. The following troubleshooting techniques may help resolve the problem

- If you are using a container, is the container runtime working? Try `docker run --gpus all ubuntu nvidia-smi` - if this doesn't work, Ollama won't be able to see your NVIDIA GPU.
- Is the uvm driver loaded? `sudo nvidia-modprobe -u`
- Try reloading the nvidia_uvm driver - `sudo rmmod nvidia_uvm` then `sudo modprobe nvidia_uvm`
- Try rebooting
- Make sure you're running the latest nvidia drivers

If none of those resolve the problem, gather additional information and file an issue:

- Set `CUDA_ERROR_LEVEL=50` and try again to get more diagnostic logs
- Check dmesg for any errors `sudo dmesg | grep -i nvrm` and `sudo dmesg | grep -i nvidia`

AMD GPU Discovery

On linux, AMD GPU access typically requires `video` and/or `render` group membership to access the `/dev/kfd` device. If permissions are not set up correctly, Ollama will detect this and report an error in the server log.

When running in a container, in some Linux distributions and container runtimes, the ollama process may be unable to access the GPU. Use `ls -lnd /dev/kfd /dev/dri /dev/dri/*` on the host system to determine the **numeric** group IDs on your system, and pass additional `--group-add ...` arguments to the container so it can access the required devices. For example, in the following output `crw-rw---- 1 0  44 226,   0 Sep 16 16:55 /dev/dri/card0` the group ID column is `44`

If you are experiencing problems getting Ollama to correctly discover or use your GPU for inference, the following may help isolate the failure.

- `AMD_LOG_LEVEL=3` Enable info log levels in the AMD HIP/ROCm libraries. This can help show more detailed error codes that can help troubleshoot problems
- `OLLAMA_DEBUG=1` During GPU discovery additional information will be reported
- Check dmesg for any errors from amdgpu or kfd drivers `sudo dmesg | grep -i amdgpu` and `sudo dmesg | grep -i kfd`

AMD Driver Version Mismatch

If your AMD GPU is not detected on Linux and the server logs contain messages like:

```
msg="failure during GPU discovery" ... error="failed to finish discovery before timeout"
msg="bootstrap discovery took" duration=30s ...
```

This typically means the system's AMD GPU driver is too old. Ollama bundles ROCm 7 linux libraries which require a compatible ROCm 7 kernel driver. If the system is running an older driver (ROCm 6.x or earlier), GPU initialization will hang during device discovery and eventually time out, causing Ollama to fall back to CPU.

To resolve this, upgrade to the ROCm v7 driver using the `amdgpu-install` utility from AMD's ROCm documentation. After upgrading, reboot and restart Ollama.

Multiple AMD GPUs

If you experience gibberish responses when models load across multiple AMD GPUs on Linux, see the following guide.

- https://rocm.docs.amd.com/projects/radeon-ryzen/en/latest/docs/install/installrad/native_linux/mgpu.html#mgpu-known-issues-and-limitations

Windows Terminal Errors

Older versions of Windows 10 (e.g., 21H1) are known to have a bug where the standard terminal program does not display control characters correctly. This can result in a long string of strings like `←[?25h←[?25l` being displayed, sometimes erroring with `The parameter is incorrect` To resolve this problem, please update to Win 10 22H1 or newer.

□□□ #1
□ □□□ □□□ 2026-10-31 13:18:39 UTC
□ □□□ □□□ 2026-10-31 13:18:39 UTC