llama.cpp Build Errors: Common Fixes for Every Platform

February 18, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** The most common build failures: Ubuntu 22.04 ships CMake 3.22 but llama.cpp needs 3.14+ (usually fine) — the real issue is missing CUDA toolkit or build-essential. On Mac, missing Xcode CLI tools prevents Metal. On Windows, missing Visual Studio Build Tools. For each: install the dependency, re-run cmake, rebuild. If the build succeeds but inference crashes, you have a CUDA architecture mismatch — rebuild with -DGGML_CUDA_ARCHITECTURES matching your GPU.

📚 **More on this topic:** llama.cpp vs Ollama vs vLLM · AMD vs NVIDIA for Local AI · Run Your First Local LLM

llama.cpp is the engine behind most local AI inference. Ollama wraps it so you never see the build process. But if you're building from source — for speed, control, or because you need features Ollama doesn't expose — the build will break at least once.

This is the fix guide. Find your error, get the fix, move on.

Before You Start

The standard build process:

```
git clone https://github.com/ggml-org/llama.cpp
cd llama.cpp
cmake -B build -DGGML_CUDA=ON    # or -DGGML_METAL=ON for Mac
cmake --build build --config Release -j $(nproc)
```

If that worked, you don't need this page. If it didn't, find your error below.

Linux Errors

“CMake Error: CMake 3.x or higher is required”

Error:

```
CMake Error at CMakeLists.txt:1:
  CMake 3.14 or higher is required. You are running version 3.10.
```

Cause: Your distro ships an old CMake. Common on Ubuntu 20.04 and some 22.04 installs.

Fix:

```
# Remove old cmake and install from Kitware repo
sudo apt remove cmake
sudo apt install -y gpg wget
wget -O - https://apt.kitware.com/keys/kitware-archive-latest.asc | sudo gpg --dearmor -o /usr/s
echo 'deb [signed-by=/usr/share/keyrings/kitware-archive-keyring.gpg] https://apt.kitware.com/ub
sudo apt update && sudo apt install cmake
```

Or use pip: `pip install cmake` and ensure `~/.local/bin` is in your PATH.

“Could NOT find CUDAToolkit”

Error:

```
CMake Error at /usr/share/cmake/Modules/FindCUDAToolkit.cmake:
  Could not find cuda_runtime.h
```

Cause: CUDA toolkit isn’t installed, or CMake can’t find it.

Fix:

```
# Install CUDA toolkit (Ubuntu/Debian)
sudo apt install nvidia-cuda-toolkit

# Or install from NVIDIA directly (more control over version):
# https://developer.nvidia.com/cuda-downloads
```

```
# Verify it's found
nvcc --version
```

If `nvcc` works but CMake still can't find it, set the path explicitly:

```
cmake -B build -DGGML_CUDA=ON -DCMAKE_CUDA_COMPILER=/usr/local/cuda/bin/nvcc
```

"No CMAKE_CXX_COMPILER could be found"

Error:

```
CMake Error: CMAKE_CXX_COMPILER not set
```

Cause: No C++ compiler installed.

Fix:

```
sudo apt install build-essential
```

This installs gcc, g++, make, and core development headers. Required for any C++ build on Debian/Ubuntu.

CUDA Architecture Mismatch

Error: Build succeeds, but running inference gives:

```
CUDA error: no kernel image is available for execution on the device
```

Cause: llama.cpp was compiled for a different GPU generation than yours. By default, CMake builds for common architectures, but it can miss older or newer cards.

Fix: Specify your GPU's compute capability:

GPU	Compute Capability
RTX 3060/3070/3080/3090	86
RTX 4060/4070/4080/4090	89

GPU	Compute Capability
RTX 5070/5080/5090	100
GTX 1080 Ti	61
Tesla P40	61

```
# Example for RTX 3090
cmake -B build -DGGML_CUDA=ON -DGGML_CUDA_ARCHITECTURES="86"

# Multiple GPUs / generations
cmake -B build -DGGML_CUDA=ON -DGGML_CUDA_ARCHITECTURES="86;89"
```

Find your GPU's compute capability: `nvidia-smi --query-gpu=compute_cap --format=csv`

AMD ROCm Build Fails

Error:

```
Could not find HIP
```

Cause: ROCm isn't installed or the environment isn't set up.

Fix:

```
# Install ROCm (see AMD docs for your distro)
# Then build with HIP backend:
cmake -B build -DGGML_HIP=ON
cmake --build build --config Release -j $(nproc)
```

ROCm requires specific kernel versions and supported GPUs. Check AMD's compatibility list. The RX 7900 XTX and RX 7900 XT are the best-supported consumer cards. See our AMD vs NVIDIA guide for details.

---

macOS Errors

Metal Backend Not Enabling

Error:

```
-- Metal: disabled
```

Or the build succeeds but inference runs on CPU.

Cause: Missing Xcode Command Line Tools. Metal support requires Apple's development headers.

Fix:

```
xcode-select --install
```

Wait for the install to complete, then rebuild:

```
cmake -B build -DGGML_METAL=ON
cmake --build build --config Release -j $(sysctl -n hw.ncpu)
```

Verify Metal is enabled in the CMake output — look for `-- Metal: enabled`.

Homebrew CMake Conflicts

Error:

```
CMake Error: The source directory does not appear to contain CMakeLists.txt
```

Or CMake behaves erratically after a Homebrew update.

Cause: Multiple CMake versions installed (system + Homebrew), or stale build cache.

Fix:

```
# Clean the build directory completely
rm -rf build
```

```
# Use Homebrew's cmake explicitly
/opt/homebrew/bin/cmake -B build -DGGML_METAL=ON
/opt/homebrew/bin/cmake --build build --config Release
```

ARM vs x86 Build Confusion

Error: Build succeeds but runs slowly, or crashes with architecture errors.

Cause: On M-series Macs, you might be building x86 via Rosetta instead of native ARM.

Fix: Check your build:

```
file build/bin/llama-cli
# Should say: Mach-O 64-bit executable arm64
# NOT: Mach-O 64-bit executable x86_64
```

If it says x86_64, you're building under Rosetta. Make sure your terminal is running natively (check in Activity Monitor) and that Homebrew is the ARM version (`/opt/homebrew/bin/brew`, not `/usr/local/bin/brew`).

Windows Errors

"CMake Error: Visual Studio not found"

Error:

```
CMake Error: CMAKE_CXX_COMPILER not set, after EnableLanguage
No CMAKE_C_COMPILER could be found
```

Cause: Visual Studio Build Tools aren't installed.

Fix: Install Visual Studio Build Tools. During installation, select "Desktop development with C++". Then build from a Developer Command Prompt or Developer PowerShell:

```
cmake -B build -DGGML_CUDA=ON
cmake --build build --config Release
```

CUDA Path Not Found

Error:

```
Could NOT find CUDAToolkit (missing: CUDA_TOOLKIT_ROOT_DIR)
```

Cause: CUDA toolkit installed but not in the system PATH.

Fix: Add CUDA to your environment variables:

```
# Check where CUDA is installed
where nvcc

# Add to PATH if missing (typical location)
$env:PATH += ";C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v12.4\bin"
```

Or set it permanently in System Properties → Environment Variables. Restart your terminal after changing PATH.

WSL2 vs Native: When Each Works Better

WSL2 wins when: You're more comfortable with Linux, you want to follow Linux build guides directly, or you need ROCm for AMD.

Native Windows wins when: You want maximum GPU performance (WSL2 adds ~5-10% overhead) or you're using CUDA and want the simplest path.

For WSL2 builds, install the NVIDIA CUDA toolkit inside WSL2, not on the Windows side:

```
# Inside WSL2 Ubuntu
sudo apt install nvidia-cuda-toolkit
```

The GPU driver comes from Windows — don't install NVIDIA drivers inside WSL2.

Common Across All Platforms

“fatal: not a git repository” or Missing ggml

Error:

```
CMake Error: The source directory "ggml" does not contain a CMakeLists.txt
```

Cause: Submodules weren’t cloned. llama.cpp depends on the ggml library as a git submodule.

Fix:

```
git submodule update --init --recursive
```

Or re-clone with submodules:

```
git clone --recurse-submodules https://github.com/ggml-org/llama.cpp
```

“undefined reference to…” Link Errors

Error:

```
undefined reference to `cublasCreate_v2'
undefined reference to `ggml_cuda_init'
```

Cause: Library mismatch. The compiler found headers but not the matching libraries. Common when CUDA toolkit version doesn’t match what CMake found.

Fix: Clean build and specify paths explicitly:

```
rm -rf build
cmake -B build -DGGML_CUDA=ON -DCMAKE_CUDA_COMPILER=/usr/local/cuda/bin/nvcc
cmake --build build --config Release -j $(nproc)
```

If you have multiple CUDA versions installed, ensure consistency: `nvcc --version` should match the CUDA libs you’re building against.

Out of Disk Space

Error:

```
No space left on device
```

Cause: The build generates several GB of intermediate files. Debug builds are especially large.

Fix: Release builds are smaller. Make sure you're building with `--config Release`:

```
cmake --build build --config Release
```

Or free space. The build directory itself can be 2-5GB. Models are 2-50GB each.

Build Succeeds, Inference Crashes

Symptoms: Binary exists, starts running, then CUDA error or segfault.

Check these in order:

1. **CUDA arch mismatch** — rebuild with `-DGGML_CUDA_ARCHITECTURES` set to your GPU (see table above)
2. **Driver too old** — run `nvidia-smi` to check driver version. CUDA 12.x needs driver 525+.
3. **Model file corrupt** — re-download the GGUF file. Partial downloads cause silent corruption.
4. **VRAM too low** — the model doesn't fit. See CUDA out of memory fix.

The Easy Alternative

If you don't need to build from source, Ollama bundles llama.cpp with pre-built binaries for every platform. One command, no build process:

```
curl -fsSL https://ollama.com/install.sh | sh
ollama run qwen3:8b
```

LM Studio does the same with a GUI. Both use llama.cpp internally — you get the same inference engine without touching a compiler.

Build from source when you need: bleeding-edge features, custom CUDA architectures, specific compile flags, or maximum performance tuning. For everything else, the pre-built options work.
