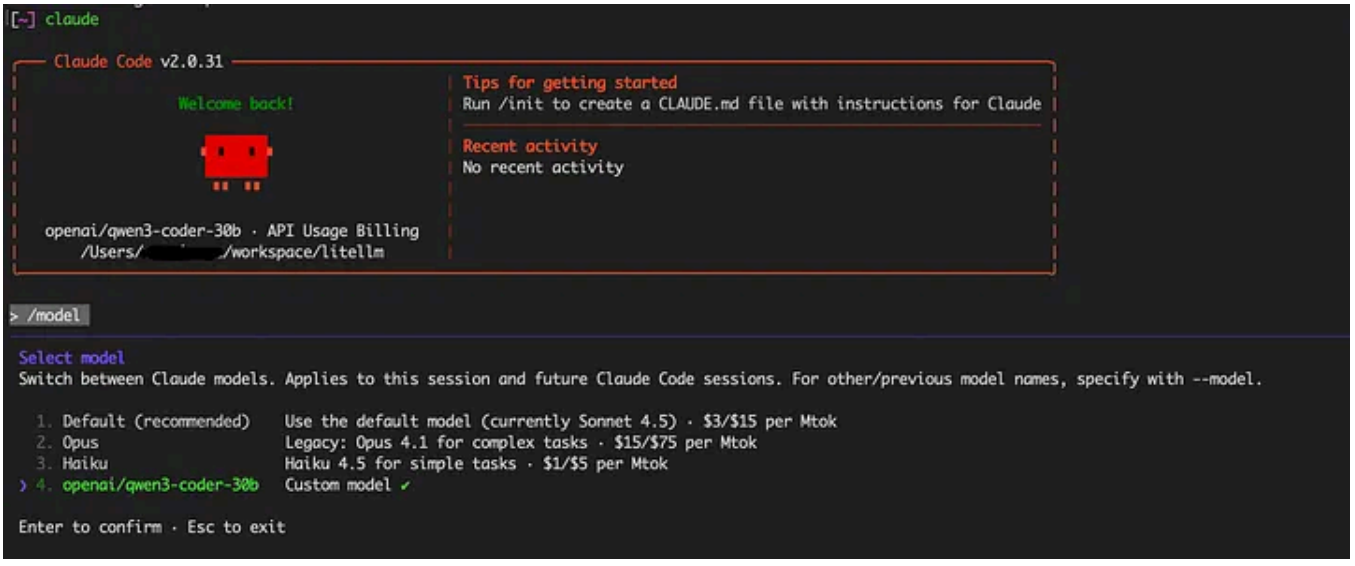
# Setting Up Claude Code Locally with a Powerful Open-Source Model: A Step-by-Step Guide for Mac Users

5 min read · Nov 2, 2025

Luong NGUYEN Follow






Claude Code with local model Qwen3-coder-30b

Hey there, fellow coders and AI enthusiasts! If you're like me, you love the idea of having an AI-powered coding assistant right at your fingertips, but without relying on cloud services that can rack up costs or raise privacy concerns. That's where running Claude Code locally comes in. In this blog post, I'll walk you through how to set up Claude Code — a nifty CLI tool from Anthropic for AI-assisted coding — using a high-performance open-source model like Qwen3 Coder 30B, all running on your local machine.

This setup is tailored for a MacBook M1 Max with 32GB of unified memory, but it should work on similar Apple Silicon setups. We'll use LM Studio to host the model,

LiteLLM as a proxy to bridge the APIs, and Claude Code for the actual coding workflow. By the end, you'll have a fully local, cost-free coding powerhouse that rivals cloud-based options. Let's dive in!

## Why Go Local with Claude Code?

Before we get our hands dirty, a quick rundown on why this is awesome:

- **Privacy and Control**: Your code stays on your machine — no data sent to external servers.
- **Cost Savings**: No API fees; everything runs offline.
- **Performance**: On an M1 Max, expect snappy responses (10–30 tokens/sec) with models optimized for coding tasks.
- **Flexibility**: Use it in your terminal, IDE, or even integrate with VS Code.

I chose Qwen3 Coder 30B because it's a beast for coding — strong in generation, debugging, and reasoning — while fitting comfortably in 32GB RAM when quantized. If you're on a different setup, you could swap it for alternatives like Codestral 22B or DeepSeek R1 33B.

## Prerequisites

Make sure you've got these ready:

**Hardware**: MacBook M1 Max (or similar) with 32GB unified memory.

**Software**:

- LM Studio (download from lmstudio.ai).
- Docker (from docker.com — essential for LiteLLM).
- Node.js (v20+; install via `brew install node` if you have Homebrew).
- Basic terminal skills — we'll be using commands here and there.
- The Qwen3 Coder 30B model: Search for "Qwen/Qwen3-Coder-30B-A3B-Instruct-GGUF" in LM Studio's model hub and download the 4-bit quantized version (Q4_K_M) for efficiency (~17GB size).

## Step 1: Set Up the Local Model in LM Studio

LM Studio is your gateway to running large language models locally with GPU acceleration on Apple Silicon.

1. Open LM Studio and load the Qwen3 Coder 30B model (use Q4_K_M quantization to keep memory usage low).
2. Head to the “Local Inference Server” tab (look for the gear icon).
3. Configure the server:

- Port: Stick with 1234 (default).
- Enable CORS if you plan on cross-origin requests (optional but handy).
- Context length: Start with 32K tokens to save memory; bump it up to 128K for bigger projects if needed.
- GPU offload: Maximize layers to Metal for that M1 Max speed boost.

4. Hit “Start Server.” This exposes an OpenAI-compatible API at `http://localhost:1234/v1/chat/completions`.

5. Test it out in your terminal: `curl http://localhost:1234/v1/models`. If it responds with model info, you're golden!

Pro Tip: Monitor memory in Activity Monitor. If things get tight, reduce context length or quit background apps.

### Step 2: Install Claude Code

Claude Code is the star of the show — it’s a CLI tool that lets you interact with AI for coding tasks like implementing functions, debugging, or refactoring entire files.

1. In your terminal, run: `npm install -g @anthropic-ai/claude-code`.
2. Verify the install: Type `claude` —it might ask for an Anthropic key, but we'll override that soon.

That’s it for installation. Now, we need to trick it into using our local model instead of Anthropic’s cloud API.

### Step 3: Set Up LiteLLM as a Proxy

LiteLLM is the magic bridge that translates Anthropic-style requests from Claude Code to our local OpenAI-compatible server in LM Studio. We'll run it via Docker for a clean, persistent setup.

1. Create a new directory: `mkdir -p litellm && cd litellm`.
2. Create a `.env` file with this content:

```
LITELLM_MASTER_KEY="sk-1234"  # This is your arbitrary auth key—keep it secure!
```

3. Create `config.yaml`:

```
model_list:
  - model_name: "anthropic/*"  # Maps all Anthropic models to your local one
    litellm_params:
      model: "openai/qwen3-coder-30b"  # Custom name for your model
      api_base: "http://host.docker.internal:1234/v1"  # Points to LM Studio
      api_key: "lm-studio"  # Dummy key (not actually needed)
      max_tokens: 65536
      repetition_penalty: 1.05
      temperature: 0.7
      top_k: 20
      top_p: 0.8
```

4. Create `docker-compose.yml` (this sets up LiteLLM with a PostgreSQL database for config storage):

```
services:
  litellm:
    image: ghcr.io/berriai/litellm:main-stable
    command: ["--config=/app/config.yaml"]
    container_name: litellm
    restart: unless-stopped
    volumes:
      - ./config.yaml:/app/config.yaml
    ports:
      - "4000:4000"
    env_file:
      - .env
```

```
    depends_on:
      - db
    healthcheck:
      test: ["CMD-SHELL", "wget --no-verbose --tries=1 http://localhost:4000
      interval: 30s
      timeout: 10s
      retries: 3
      start_period: 40s

  db:
    image: postgres:16
    restart: always
    container_name: litellm_db
    environment:
      POSTGRES_DB: litellm
      POSTGRES_USER: llmproxy
      POSTGRES_PASSWORD: dbpassword9090
    ports:
      - "5432:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -d litellm -U llmproxy"]
      interval: 1s
      timeout: 5s
      retries: 10

volumes:
  postgres_data:
    name: litellm_postgres_data
```

5. Start it up: `docker compose up -d`.

6. Verify the proxy:

Run `curl -H "Authorization: Bearer sk-1234" http://localhost:4000/health`

You should see something like `{"healthy": true}`. If you get a 401 error without the header, that's expected—auth is enabled for security.

If the health check fails, peek at the logs with `docker logs litellm` or restart with `docker compose down && docker compose up -d`.

### Step 4: Configure Claude Code to Use the Local Proxy

Now, link everything together by setting environment variables. These tell Claude Code to route through LiteLLM.

Add these to your `~/.zshrc` or `~/.bashrc` (then `source` the file to apply):

```
export ANTHROPIC_AUTH_TOKEN="sk-1234"  # Matches your LiteLLM key
export ANTHROPIC_BASE_URL="http://localhost:4000"
export ANTHROPIC_MODEL="openai/qwen3-coder-30b"
export ANTHROPIC_SMALL_FAST_MODEL="openai/qwen3-coder-30b"
export CLAUDE_CODE_DISABLE_NONESSENTIAL_TRAFFIC=1  # Optional: No telemetry
```

### Step 5: Launch and Test Claude Code

1. Navigate to a project directory and run `claude`.
2. Inside the CLI, type `/model` to confirm it's using your custom Qwen3 setup.
3. Start prompting! For example: “Implement a Python function to sort a list.” Watch as it generates code using your local model.
4. Bonus: For VS Code fans, install the Claude Code extension (if available) and add the env vars to your `settings.json` under `"terminal.integrated.env.osx"`.

### Troubleshooting Common Issues

- **Memory Swaps**: If your Mac starts swapping, lower the context length in LM Studio or try a higher quantization like Q6_K (if memory allows).
- **Tool Calling Glitches**: Local models might need tweaks — experiment with custom Jinja templates in LM Studio for better parsing.
- **API Mismatches**: If features like file edits fail, fall back to directing Claude Code straight to LM Studio’s endpoint (skip LiteLLM), though you might lose some compatibility.
- **Slower Performance?** Qwen3 is efficient, but for speed demons, consider smaller models like Qwen 2.5 Coder 7B.
- **Web Version Note**: Anthropic has a browser-based Claude Code beta, but it doesn’t support local models — stick to the CLI for this setup.

### Final Thoughts

There you have it — a complete, local Claude Code setup that’s powerful, private, and perfect for coding marathons. I’ve been using this for everything from quick scripts to debugging complex projects, and it’s a game-changer. If you run into snags or

have tweaks to share, drop a comment below. Happy coding, and may your bugs be few!

*This guide is based on setups tested on macOS with M1 Max. Tools and versions evolve, so check official docs for updates.*

Vibe Coding | Claude Code | Software Development



Follow

## Written by Luong NGUYEN

11 followers · 1 following

See the world in your own way!

## Responses (3)

Bgerby

What are your thoughts?



*石涵*

14 hours ago



The biggest problem with this solution is the context window size issue. Long-term programming or debugging can easily cause the context to exceed the model's limits, and Claude Code does not offer a solution. You might want to check out OpenCode... more

1 Reply

Reski Rukmantiyo
16 hours ago

Thanks for this guide. Quick question
- will claude like subagents and skills works?
- how is the quality? Suppose I'm using qwen 3 420b from openrouter as my litellm backend?

Reply

mufu
1 day ago

A quick question, do you think this would function well on an M4 macbook air?

1 reply Reply

## More from Luong NGUYEN

Luong NGUYEN

### Discovering Claude Code: Slash Commands

This is Part 1 of my series on discovering and learning Claude Code, where I explore the powerful features that make AI-assisted...

2d ago

Luong NGUYEN

## Demystifying Your Network Traffic on macOS: A Docker-Based Approach with MMT

The AI Analysis Paradox: Tools Without Data

May 2 50

Luong NGUYEN

## Building an Intelligent Email Phishing Detector with LLMs and LangGraph

Phishing emails remain a persistent threat, constantly evolving to trick users into revealing sensitive information or clicking malicious...

Apr 21 1

See all from Luong NGUYEN

## Recommended from Medium

In AI Software Engineer by Joe Njenga

## Anthropic Just Solved AI Agent Bloat—150K Tokens Down to 2K (Code Execution With MCP)

Anthropic just released smartest way to build scalable AI agents, cutting token use by 98%, shift from tool calling to MCP code execution

4d ago 417 33

ZIRU

## How to Build Software That Actually Works: The Claude Code + Speckit Method

A practical workflow that turned my chaotic dev process into a systematic approach

6d ago 46 1

Yogeshwar Tanwar

## The $699 Mac Mini Server That Replaced My AWS Bill: The Devil's in the Details (Part 2)

Three months in, and I'm still waiting for the disaster. The catastrophic failure that would send me crawling back to AWS at 3 AM with my...

Oct 13 248

Bhavyansh

## My Favorite 8 CLI Tools for Everyday Development (2025 Edition)

Replace your old Unix tools with modern alternatives that actually respect your time

5d ago 53 5

Rami Krispin

## Setting Up OpenCode with Local Models

A completely open and free alternative to Claude Code

6d ago 2

IcePanel

## Top MCP tools for software architects

10 MCP tools to keep an eye on for software architecture.

5d ago 80