

Run Frontier Code *Offline.*

The complete guide to Qwen 3.8-27B: what it can really do, how to install it in LM Studio, how to make it fast, and how to wire it to GitHub and the internet as a working coding agent on your own laptop.

61.7

SWE-BENCH PRO, ABOVE
CLAUDE OPUS 4.6 MAX ON
QWEN'S OWN TABLE

17 GB

ONE FILE. RUNS ON A 24 GB
LAPTOP

\$0

PER TOKEN, FOREVER. APACHE
2.0 LICENCE

Why this model, and why now

For three years the rule was simple: the smart models live in the cloud, and you rent them by the token. In August 2026 Alibaba released Qwen 3.8-27B under an Apache 2.0 licence, and the rule broke. It is a 27-billion-parameter model that fits in a 17 GB file, runs on a laptop, and on Alibaba's own coding tests **scores above models you are currently paying for.**

This guide is not a review. It is the exact path from an empty laptop to a coding agent that reads your repos, searches the web and opens pull requests, with the model running on hardware you own. Four stages, in order.

STAGE 1

Install

LM Studio, plus the right version of the model for the memory you have.

STAGE 2

Tune

Five settings that take it from "slow" to faster than most cloud APIs.

STAGE 3

Harness

Claude Code, OpenCode or Qwen Code, all pointed at LM Studio's server.

STAGE 4

Connect

GitHub, web search and a real browser, through MCP servers.

\$0/mo

That is the running cost of the whole stack. The model, LM Studio, every harness and every connector in this guide is free. The only thing you pay for is a machine with enough memory, and there is a good chance you already own one.

One rule before you start: **do not judge this model at its default setting.** It ships with reasoning set to "xhigh", which makes it think for minutes about trivial questions. Page 8 fixes that in one click. Most of what you will read online about it being slow comes from people who never found that click.

What you are actually downloading

SIZE

27B, dense

Every parameter works on every token. No expert routing, so quality is consistent and it shrinks well to 4-bit.

CONTEXT

262,144 tokens

Native. Roughly 200,000 words, or a medium-sized codebase, in one conversation.

VISION

Sees images

Native image and video input. Paste a screenshot of a broken layout and ask it to fix the CSS.

SPEED

Drafter built in

A multi-token prediction head ships inside the file. One setting makes it up to 1.8x faster.

Why it behaves so well on a laptop

The architecture is the interesting part. Of its 64 layers, 48 are "Gated DeltaNet" layers that keep a fixed-size memory no matter how long the conversation gets. Only 16 are classic attention layers that grow a cache with every token. In plain English: **the part of the model that eats memory as the context grows is a quarter of the size it would be in a conventional design.**

The numbers: those 16 attention layers store about 64 KB per token at full precision. A 32,000-token conversation costs around 2 GB of cache, and about half a gigabyte if you quantise the cache to 4-bit. That is why a 24 GB machine can hold the model and a serious amount of context at the same time, and why generation speed barely drops as the context fills. One test rig measured it flat between 80,000 and 260,000 tokens.

Three things it does that matter to you

- **Thinking with a dial.** Reasoning is on by default at "xhigh". You can set it to medium, low, or off per request. This single setting controls most of your waiting time.
- **Tool calling, trained in.** It was built for "long-horizon agentic tasks": reading files, running commands, reacting to errors. That is what makes it a coding agent rather than a chatbot.
- **Preserved thinking.** It keeps its reasoning across turns of a conversation, which helps on multi-step work and costs context. Turn it off for quick chat.

The recommended sampling settings, straight from the model card.

Thinking on: temperature 1.0, top_p 0.95, top_k 20, min_p 0, presence penalty 0.

Thinking off: temperature 0.7, top_p 0.80, top_k 20, presence penalty 1.5. That higher presence penalty is not optional: it is what stops the model looping when it has no reasoning trace to steady it.

For agent work, allow long outputs. Qwen recommends up to 262,144 tokens of reasoning and 131,072 of answer for agentic tasks.

Sources: Qwen/Qwen3.8-27B model card (Hugging Face, Aug 2026); vramcalculator.com speed analysis; kingy.ai quantisation guide.

How good is it, honestly

These are the headline numbers from Qwen's model card, side by side with the previous local model and the closed model Qwen chose to compare against. Higher is better on every row.

BENCHMARK	WHAT IT MEASURES	QWEN 3.8-27B	QWEN 3.6-27B	OPUS 4.6 MAX
Terminal-Bench 2.1	Agentic work in a real terminal	73.0	63.4	78.2
SWE-bench Pro	Fixing real bugs in real repos	61.7	53.5	53.4
LiveCodeBench v6	Competitive programming	90.3	83.9	88.8
GPQA Diamond	PhD-level science questions	89.2	87.8	91.3
Humanity's Last Exam	Expert questions, all fields	30.8	24.0	40.0
OSWorld-Verified	Operating a computer by screen	84.3	63.9	72.7
IFBench	Following instructions exactly	79.5	69.1	62.5
Vision2Web	Screenshot to working web page	62.9	45.0	n/a

Source: Qwen3.8-27B model card, Hugging Face, August 2026. SWE-bench Pro and Vision2Web were run inside the Claude Code harness. Orange means the 27B scored highest of the three.

Read the table like a professional

- **These are Alibaba's numbers.** No independent lab has reproduced them yet. Treat them as a ceiling, not a promise.
- **Independent scores are lower.** Artificial Analysis' Intelligence Index (v4.3) gives it 34. GLM-5.3-Flash, a 320B model, gets 42. Frontier closed models sit in the low 50s. BenchLM ranks it 39th of 232 models at 64.7 out of 100 (15 Sep 2026).
- **The 4-bit file loses almost nothing.** Quesma tested the quantisations: on GPQA Diamond the 4-bit file scored about 93 against 94 at full precision; on Terminal-Bench 2.1, 77 against 78. The 2-bit file slipped to 74. The 1-bit file collapsed to random guessing.

What it is good at, from people who ran it as an agent: implementing a well-specified function, writing tests, tracing a bug through a few files, refactoring a module, writing documentation. It follows instructions in a project's markdown files consistently, which is what makes agent mode work at all.

What it struggles with: sprawling multi-file architecture changes, subtle reasoning about unfamiliar library internals, and long autonomous runs where one early mistake compounds.

The play: give it the 80% of coding that is routine. Keep a cloud model for the 20% that is not.

Sources: artificialanalysis.ai open-source comparison; benchlm.ai; quesma.com quantisation benchmark; zachrattnr.com Mac agentic coding write-up.

What your hardware will actually do

Generation speed is a memory bandwidth problem, not a compute problem. A 17 GB model has to stream through memory once per token, so the machine with the fastest memory wins. Then the built-in drafter (MTP, page 8) roughly doubles that number on GPUs. All figures below are measured, 4-bit files, tokens per second while generating, except the one row marked est.

MACHINE	MEMORY	BASELINE	WITH MTP	VERDICT
DESKTOP GPUS				
RTX 3090	24 GB	31 to 40	41 to 63	The value pick. Cheap used, 24 GB is enough.
RTX 4090	24 GB	46 to 48	65 to 76	Comfortable at 64k context.
RTX 5090	32 GB	61 to 75	134 to 137	Fastest single card. Street price ~\$4,600+ in Sep 2026 against a \$1,999 MSRP.
Radeon RX 7900 XTX	24 GB	31	44	AMD works. Expect fewer tuning guides.
LAPTOPS AND UNIFIED MEMORY				
MacBook Pro M3 Pro The author's own laptop (14-inch, 2023)	18 GB	10 to 12 est.	No room	Q4 will not fit. Use the 2-bit UD-Q2_K_XL (10.7 GB) at 8k to 16k context: good for chat and single-file help, a squeeze for an agent.
MacBook Pro M5 Max	36 GB	18	31	A real laptop seat. Flash attention on.
MacBook Pro M5 Max	128 GB	15 to 31	about 1.7x	Full context fits. Simon Willison's test rig.
Mac mini M4	32 GB	5 to 6	n/a	MLX. Fine for chat, too slow for an agent.
MacBook Air M4	24 GB	about 8	n/a	It loads. You will not enjoy it.
Ryzen AI Max+ 395	128 GB	24.5	up to 36	Budget unified memory. AMD figure, then tuned.
RTX 5090 Laptop	24 GB	3090-class	3090-class	The Windows laptop pick, heat-limited.
RTX 5080 Laptop	16 GB	about 4	n/a	Spills to system RAM. Avoid.

Sources: KGP Talkie (M5 Max vs RTX 5090, llama.cpp b10793); Hardware Corner GPU sweep; sudoingX qwen38-mtp paired benchmarks; AMD day-0 blog; Context Studios hardware guide; orcarouter.ai and thomas-wiegold.com Mac tests; PCGamesN and videocardprices.com for the 5090 price. M3 Pro row: estimated from its 150 GB/s bandwidth (MacRumors) and the ~13.5 GB default GPU limit on an 18 GB Mac (quantized.uk), calibrated against the measured M4 rows.

The three speed thresholds. Under 10 tokens/s: chat only. 20: the floor for an agent you will wait for. 40 and up: it feels like a cloud model.

Mac users. MLX decodes about 40% faster than GGUF at 4-bit, but LM Studio's MTP drafter runs on GGUF. Time a 300-token answer on each, keep the faster.

Which file to download

The model comes in "quantisations": the same weights stored with fewer bits per number. Smaller files run on smaller machines and lose a little precision. The quality column is the share of tokens where the compressed file makes the same top choice as the full-precision original.

FILE	SIZE	QUALITY KEPT	MIN. MEMORY	USE IT WHEN
GGUF (WINDOWS, LINUX, MAC; SUPPORTS THE MTP DRAFTER)				
IQ2_S	11.1 GB	87%	12 to 16 GB	Emergency only. Expect mistakes.
IQ3_S	13.8 GB	92%	16 GB	A 16 GB machine, 8k context, chat use.
IQ4_XS	16.5 GB	95%	20 GB	A tight 24 GB card that needs long context.
Q4_K_M (LM Studio pick)	16.8 GB	about 96%	24 GB	The default. No measurable loss on coding tests.
UD-Q4_K_XL (Unsloth)	17.9 GB	96%	24 GB	The best 4-bit file if you have 1 GB to spare.
Q6_K	22.4 GB	98%	32 GB	Quality with headroom on a 32 GB machine.
Q8_0	29 GB	99%	48 GB	Near-lossless. Only if memory is no object.
MLX (APPLE SILICON ONLY; FASTER DECODE, NO MTP DRAFTER YET)				
MLX 4-bit	about 16 GB	4-bit class	24 GB	The Mac default if the drafter matters less than raw decode.
MLX 8-bit	27 to 30 GB	8-bit class	48 GB	Mac Studio and 64 GB+ MacBook Pro.

Sources: lmstudio-community/Qwen3.8-27B-GGUF file list; kingy.ai quantisation table (publisher top-1 agreement vs BF16); unsloth.ai Qwen3.8 guide; orcarouter.ai MLX sizes. Quality for Q4_K_M is interpolated between IQ4_XS and UD-Q4_K_XL and matches Quesma's coding results.

Buy memory,
not bits.

Between Q4 and Q6 the quality difference is inside the noise on real tasks. The context headroom is not. On a 24 GB machine take Q4_K_M and spend the other 6 GB on context. On 16 GB, take IQ3_S and accept that this is a chat machine, not an agent machine.

The memory maths, measured

Total memory is weights, plus the context cache, plus about 1 GB of working buffers, plus 1 to 2 GB if the drafter is on, plus whatever your operating system keeps. Measured on an RTX 5090 with the Q4_K_M file and a 4-bit cache: **4k context 16.5 GB · 32k 18.4 GB · 64k 20.5 GB · 128k 21.6 GB · 256k 27.1 GB**. A 24 GB machine therefore tops out around 64k to 96k of usable context, and that is plenty for a coding agent.

Source: KGP Talkie llama.cpp speed-settings study, RTX 5090, September 2026.

Install it in LM Studio, step by step

LM Studio is free for personal and work use, runs on Mac (Apple silicon, macOS 14+), Windows (x64 with AVX2, or ARM) and Linux (Applmage), and needs version 0.4.22 or newer for the drafter. The current release is 0.4.24 (9 Sep 2026).

1 Download LM Studio

lmstudio.ai, then open it once. 16 GB of RAM is the stated minimum for the app; for this model you want 24 GB or more.

2 Check your memory before you download

Mac: Apple menu → About This Mac. Windows: Task Manager → Performance → GPU (dedicated memory). Match it to the table on page 6.

3 Search the catalogue

Discover tab → search `qwen3.8-27b` → pick the staff pick `qwen/qwen3.8-27b`. GGUF for everyone; MLX as a second copy on a Mac.

4 Switch to Power User mode

Bottom-left of the window. This reveals the load settings and inference settings you are about to change.

5 Load with these settings

Context length 32,768 to start. GPU offload: maximum. Flash Attention: on. K cache and V cache: q8_0 (q4_0 if memory is tight). Keep model in memory: on. Then My Models → gear icon → save as default.

6 Set the sampling

Temperature 1.0, top_p 0.95, top_k 20, min_p 0 while thinking is on. If you ever turn thinking off: temperature 0.7, top_p 0.8, presence penalty 1.5.

7 Turn reasoning down to Medium

In the chat sidebar, find the Reasoning setting and move it off "xhigh". This is the most important click in the guide. If your file only shows On and Off, see page 10.

8 Test it properly

Paste a 200-line file and ask for the bug. Watch the tokens-per-second readout under the reply. Then drag in a screenshot and ask what it sees: the vision path is part of the same file.

OR DO IT FROM THE TERMINAL WITH LMS, LM STUDIO'S CLI

```
# one-time: put lms on your PATH (Mac/Linux; Windows: %USERPROFILE%\lmstudio\bin\lms.exe bootstrap)
~/.lmstudio/bin/lms bootstrap

# download, load with your settings, list what is loaded
lms get qwen/qwen3.8-27b
lms load qwen/qwen3.8-27b --gpu max --context-length 32768
lms ps
```

Sources: lmstudio.ai system requirements, changelog 0.4.22 and 0.4.24, CLI docs, "free for work" announcement; Qwen model card sampling parameters.

Make it fast: five levers, ranked by effect

01 Reasoning effort

THE BIGGEST LEVER

At "xhigh" the model thinks at maximum depth about everything. Simon Willison's SVG test: 21 minutes and 22,276 thinking tokens for 3,223 tokens of answer; the same prompt with reasoning off took 137 seconds. On an RTX 5090, time to the first visible answer went from 1.44 s at xhigh to 0.21 s off, and decode from 129 to 151 tokens/s. **Rule: Medium for coding, Low for chat, xhigh only for the one bug that has beaten you all day.**

02 The MTP drafter

UP TO 1.8X, LOSSLESS

The file carries a small "draft head" that guesses the next few tokens; the full model checks them in one pass and keeps only what it would have written anyway, so output is identical. LM Studio 0.4.22+ (llama.cpp engine 2.29.1+): chat sidebar → Speculative Decoding → the built-in MTP drafter (DFlash and DSpark drafters also work). Measured: RTX 4090 47.7 to 76.3 tokens/s, RTX 5090 73.6 to 133.6, M5 Max 18.2 to 31.1. Costs 1 to 2 GB. Three draft tokens is the sweet spot on Mac and GPU alike.

03 Quantise the cache

FASTER AND SMALLER

Setting the K and V cache to q4_0 took an RTX 5090 from 125.5 to 136.7 tokens/s **and** freed 1.4 GB, with no measured quality loss on that test. q8_0 is the cautious middle. Needs Flash Attention on.

04 Flash Attention on

MANDATORY ON A MAC

Off, it costs 2.3 GB of VRAM and a fifth of your prompt-processing speed on a GPU. On a 36 GB MacBook Pro, off pushed the working buffer from 257 MB to 1.66 GB, forced 3.2 GB of swap and stalled one run in three: a single 256-token reply took 952 seconds while the log showed nothing wrong.

05 Size the context to the job

FREE UNTIL IT IS NOT

A big window is nearly free while prompts are short. Filling it is not: on one RTX 5090 a full 128k prompt took 78 seconds to the first token, a full 256k prompt 757 seconds. Use 32k for chat, 64k for an agent, 128k+ only on a 32 GB+ machine when the task needs the whole repo in view.

RUNNING LLAMA.CPP DIRECTLY? THE TUNED COMMAND (LM STUDIO 0.4.24 ALSO ACCEPTS THESE AS ARGUMENT OVERRIDES)

```
llama-server -m Qwen3.8-27B-Q4_K_M.gguf -ngl 999 -fa on --jinja -c 65536 \
--cache-type-k q4_0 --cache-type-v q4_0 --spec-type draft-mtp --spec-draft-n-max 3 \
--reasoning-effort medium
```

Sources: simonwillison.net (16 Aug 2026); KGP Talkie speed-settings study and M5 Max vs RTX 5090 test; sudoingX/qwen38-mtp; LM Studio changelog 0.4.22 and 0.4.24; llama.cpp PR #22673.

The harness, explained

A model is a text predictor. A harness is the loop around it: it reads your files, runs commands, calls tools, checks the result and asks the model again. Claude Code is a harness. The model inside it is replaceable. **The loop is what turns a model into an agent.**

LM Studio makes this easy because it serves your local model through three doors, and every serious coding tool already knows how to walk through one of them. Start the server from the Developer tab, or from the terminal, and the doors are open on port 1234.

DOOR	ADDRESS	WHO WALKS THROUGH IT
Anthropic-compatible	http://localhost:1234/v1/messages	Claude Code, and anything built for Claude's API. Streaming and tool calls work out of the box.
OpenAI-compatible	http://localhost:1234/v1	OpenCode, Qwen Code, Codex CLI, Cline, Continue, Copilot Chat. Chat Completions and Responses endpoints.
LM Studio native	http://localhost:1234/api/v1/chat	Your own scripts. Accepts an "integrations" list so MCP servers run inside LM Studio.

START THE SERVER AND PROVE IT IS UP

```
lms load qwen/qwen3.8-27b --gpu max --context-length 65536
lms server start --port 1234
curl http://localhost:1234/v1/models # should list qwen/qwen3.8-27b
```

Context is the setting that breaks agents. Every harness sends a long system prompt plus your files before the model says a word. LM Studio's own guidance for Claude Code is at least 25,000 tokens; OpenCode needs 16,000 as a floor. Load at 64k for agent work and you will never see a mysterious "tool call failed".

Pick one harness. You can change later.

OPTION A

Claude Code

The most polished loop. Free CLI; pointed at LM Studio it needs no Claude subscription. Best if you already use it.

OPTION B

OpenCode

Open source, terminal UI, works with any provider. Best if you want the whole stack open.

OPTION C

Qwen Code

Built by the Qwen team for Qwen models, so it speaks their tool-call format natively. Free; its cloud free tier ended in April 2026, so run it on LM Studio.

Sources: lmstudio.ai developer docs (Anthropic compatibility, OpenAI compatibility, MCP via API), LM Studio "Use your LM Studio models in Claude Code" post; opencode.ai provider docs; QwenLM/qwen-code docs.

Option A: Claude Code on your local model

Needs LM Studio 0.4.1 or newer. Install the CLI, point two environment variables at LM Studio, name the model. That is the whole integration.

INSTALL (PICK ONE)

```
curl -fsSL https://claude.ai/install.sh | bash      # Mac, Linux, WSL
irm https://claude.ai/install.ps1 | iex           # Windows PowerShell
brew install --cask claude-code                   # Homebrew
```

RUN IT AGAINST LM STUDIO

```
lms load qwen/qwen3.8-27b --gpu max --context-length 65536
lms server start --port 1234
export ANTHROPIC_BASE_URL=http://localhost:1234
export ANTHROPIC_AUTH_TOKEN=lmstudio
cd /path/to/your/project
claude --model qwen/qwen3.8-27b
```

Windows: replace the two `export` lines with `setx ANTHROPIC_BASE_URL http://localhost:1234` and `setx ANTHROPIC_AUTH_TOKEN lmstudio`, then open a new terminal.

What you will see: the normal Claude Code prompt, with your local model named in the header. Permission prompts, slash commands and git all work as usual. Only the model changed.

Option D: LM Studio's own chat as the harness

For research rather than editing a repo, you do not need a terminal tool at all. LM Studio's chat attaches PDFs, Word files and text, uses MCP tools (page 13) with a confirmation before each call, and reads images. Plan the build and read the docs here, then hand the coding to a harness.

Set the agent's reasoning once, before you launch. APPLIES TO A, B AND C

- **Harnesses rarely pass an effort level.** Claude Code, OpenCode and Qwen Code send messages and tools, not a reasoning dial. Whatever default LM Studio holds for the model is what the agent gets. Left at xhigh, every file read comes with a minute of thinking.
- **So set the default.** Load the model, set Reasoning to Medium in the sidebar, and save it as the model's default settings. The staff-pick build (qwen/qwen3.8-27b) declares the low / medium / xhigh levels.
- **Scripts can override per call.** The OpenAI-style endpoint accepts `"reasoning_effort": "low"` in the request body, which is how you would run a cheap batch job.

If your file only shows Reasoning On / Off. Some third-party GGUFs (Unsloth's, at the time of writing) load without the effort levels, and LM Studio warns "Reasoning setting 'high' is not supported". Two fixes: use the staff-pick catalogue build, or wrap the file as a virtual model with a `reasoningEffort` custom field in its model.yaml, as documented in Unsloth's Hugging Face discussion #63 on the Qwen3.8-27B-GGUF repo.

Sources: lmstudio.ai/blog/claudecode; code.claude.com/quickstart; [lmstudio.ai/docs \(chat with documents, MCP\)](https://lmstudio.ai/docs/chat-with-documents-mcp); [lmstudio-bug-tracker issue #2284](https://lmstudio-bug-tracker.github.io/issue/2284); [unsloth/Qwen3.8-27B-GGUF discussion #63](https://unsloth.ai/Qwen3.8-27B-GGUF/discussion/63).

Option B: OpenCode

INSTALL (PICK ONE)

```
curl -fsSL https://opencode.ai/install | bash          # Mac, Linux
npm install -g opencode-ai                          # any platform with Node
brew install anomalyco/tap/opencode                  # Homebrew (Windows: scoop install opencode)
```

OPENCODE.JSON IN YOUR PROJECT (OR ~/.CONFIG/OPENCODE/)

```
{
  "$schema": "https://opencode.ai/config.json",
  "provider": {
    "lmstudio": {
      "npm": "@ai-sdk/openai-compatible",
      "name": "LM Studio (local)",
      "options": { "baseUrl": "http://127.0.0.1:1234/v1" },
      "models": { "qwen/qwen3.8-27b": { "name": "Qwen 3.8 27B (local)" } }
    }
  }
}
```

Run `opencode` in the project, type `/models` and pick the local entry. If tool calls misbehave, the context is too small: reload the model at 32k or more.

Option C: Qwen Code

The Qwen team's own terminal agent, so it speaks the model's tool-call format natively. Needs Node.js 22 or newer. Point it at LM Studio through the standard OpenAI-style variables, or make it permanent in its settings file.

INSTALL AND RUN

```
npm install -g @qwen-code/qwen-code@latest
export OPENAI_BASE_URL=http://localhost:1234/v1
export OPENAI_API_KEY=lmstudio
export OPENAI_MODEL=qwen/qwen3.8-27b
cd /path/to/your/project && qwen          # or run /auth inside and choose a custom OpenAI-compatible provider
```

OR PERMANENTLY, IN ~/.QWEN/SETTINGS.JSON

```
{
  "modelProviders": { "openai": [
    { "id": "qwen/qwen3.8-27b", "name": "Qwen 3.8 27B (LM Studio)",
      "envKey": "OPENAI_API_KEY", "baseUrl": "http://localhost:1234/v1" } ] },
  "security": { "auth": { "selectedType": "openai" } },
  "model": { "name": "qwen/qwen3.8-27b" }
}
```

Sources: opencode.ai docs (install, providers); QwenLM/qwen-code README and configuration docs (Qwen OAuth free tier discontinued 15 Apr 2026).

Connect it to GitHub

There are two ways to give the agent GitHub. Do the first today. Add the second when you want the model to read issues, review pull requests and check Actions logs as tools rather than by typing git commands.

01 Git and the GitHub CLI

START HERE

- Every harness already has a shell. Install git and the GitHub CLI and the agent can clone, branch, commit, push and open pull requests by itself.
- Mac: `brew install git gh`. Windows: `winget install Git.Git GitHub.cli`. Then `gh auth login` once; it stores the credential for git too.
- Prefer a fine-grained personal access token scoped to the repos the agent may touch: GitHub → Settings → Developer settings → Fine-grained tokens. Contents and Pull requests: read and write. Nothing else.

Your first prompt after setup: "Create a branch called fix/readme-typo, fix the typos in README.md, commit with a clear message, push, and open a pull request." Watch it run the commands. Review the PR yourself. That is the whole loop, and it just ran on a model that never left your laptop.

02 The official GitHub MCP server

WHEN YOU TRUST IT

GitHub hosts it, so there is nothing to install: point the harness at `https://api.githubcopilot.com/mcp/` with your token in the header. Default toolsets are context, repos, issues, pull_requests and users; add `actions` or `code_security` when you need them. A local Docker version (`ghcr.io/github/github-mcp-server`) exists for people who want it fully offline, and it accepts a `--read-only` flag.

ADD IT TO YOUR HARNESS (SAME SERVER, THREE SPELLINGS; LM STUDIO'S MCP.JSON IS ON PAGE 13)

```
# Claude Code
claude mcp add --transport http github https://api.githubcopilot.com/mcp/ \
  --header "Authorization: Bearer YOUR_GITHUB_PAT"

# OpenCode (opencode.json)
"mcp": { "github": { "type": "remote",
  "url": "https://api.githubcopilot.com/mcp/",
  "headers": { "Authorization":
    "Bearer YOUR_GITHUB_PAT" } } }

# Qwen Code (~/.qwen/settings.json)
"mcpServers": { "github": {
  "url": "https://api.githubcopilot.com/mcp/",
  "headers": { "Authorization":
    "Bearer YOUR_GITHUB_PAT" } } }
```

The rule for a 27B model: start read-only. It will confidently do the wrong thing sometimes, and "delete branch" is a tool call like any other. Give it write access only on repos where a bad pull request costs you five minutes, and keep the harness's confirmation prompts on for anything that pushes.

Sources: github/github-mcp-server README (remote URL, toolsets, read-only, Docker image); code.claude.com MCP docs; opencode.ai and Qwen Code MCP docs; lmstudio.ai MCP docs.

Connect it to the internet

The model knows nothing after its training cut-off and cannot open a URL. MCP servers fix that: each is a small program the harness runs for you, and the model sees it as a tool called "search" or "fetch". Install two runners once: `uv` for Python servers, Node.js 18+ for JavaScript ones.

CONNECTOR	RUNS WITH	COST	WHAT THE MODEL GETS
DuckDuckGo	<code>uvx duckduckgo-mcp-server</code>	Free, no key	Search plus page fetch, 30 searches a minute. Install first.
Fetch	<code>uvx mcp-server-fetch</code>	Free	Any URL as clean markdown, 5,000 characters per call.
Playwright	<code>npx @playwright/mcp@latest</code>	Free	A real browser: navigate, click, type, read, screenshot.
Tavily	<code>npx -y tavily-mcp@latest</code>	1,000 credits/mo free, no card	Agent-grade search with extract and crawl tools.
Brave Search	Brave's MCP server, API key	\$5 credit/mo with attribution	Free tier ended Feb 2026; then about \$3 to \$5 per 1,000 queries.

INSTALL THE RUNNERS ONCE, THEN ADD THE SERVERS

```
curl -LsSf https://astral.sh/uv/install.sh | sh # Mac, Linux
powershell -ExecutionPolicy Bypass -c "irm https://astral.sh/uv/install.ps1 | iex" # Windows
claude mcp add --transport stdio ddg -- uvx duckduckgo-mcp-server # Claude Code
claude mcp add --transport stdio fetch -- uvx mcp-server-fetch # (B and C: page 12)
```

LM STUDIO: PROGRAM TAB → INSTALL → EDIT MCP.JSON (CURSOR NOTATION, LOCAL AND REMOTE IN ONE FILE)

```
{
  "mcpServers": {
    "ddg-search": { "command": "uvx", "args": ["duckduckgo-mcp-server"] },
    "fetch": { "command": "uvx", "args": ["mcp-server-fetch"] },
    "playwright": { "command": "npx", "args": ["@playwright/mcp@latest", "--headless"] },
    "github": { "url": "https://api.githubcopilot.com/mcp/",
      "headers": { "Authorization": "Bearer YOUR_GITHUB_PAT" } }
  }
}
```

Three safety rules. Never install an MCP server you do not trust: it runs code on your machine with your permissions. Keep tool confirmations on. Treat web pages as data, not instructions: a page can carry text that tells the model to act, so keep GitHub read-only while browsing is on.

The first test. Ask: "Search for the current stable version of Next.js and give me the release date with the source URL." A working setup answers in under a minute with a real link. A broken one invents a version or says it cannot browse. Both tell you what to fix.

Sources: [nickclyde/duckduckgo-mcp-server](#); [modelcontextprotocol/servers fetch](#); [microsoft/playwright-mcp](#); [docs.tavily.com credits page](#); Brave Search API pricing and [implicator.ai](#) on the February 2026 change; [docs.astral.sh uv install](#).

A working session, and what to do when it breaks

1 Load for agent work

64k context, Reasoning Medium, MTP drafter on, cache q8_0. Note the tokens/s.

2 Open the harness in the repo

`cd project && claude` (or `opencode`, or `qwen`).

3 Cheap first prompt

"Read the README and tell me how this project is structured." It proves the loop works before you spend real time.

4 One scoped task

One function, one bug, or one test file. Name the file. Say what "done" looks like.

5 Make it run the tests

"Run the test suite and fix anything you broke." Feedback from a real terminal is where this model shines.

6 Ship on a branch

"Commit on a new branch, push, and open a PR." Then read the diff yourself. Always.

The context budget. On a Mac the slow part is not writing tokens, it is reading them: a 30,000-token prompt can take tens of seconds before the first word. Start a fresh session per task, and keep giant logs out of the conversation.

When to go to the cloud. The moment a task spans five or more files, or depends on a library the model keeps guessing wrong, hand it to a frontier model and come back. Local for the routine 80%, cloud for the hard 20%, is the setup that actually saves money.

Troubleshooting

SYMPTOM	CAUSE	FIX
Thinks for minutes about "hello"	Reasoning at xhigh	Sidebar → Reasoning → Medium or Low. Save as default.
Tool calls fail or come out garbled	Context too small for the harness	Reload at 32k or more (Claude Code wants 25k minimum).
"Reasoning setting 'high' is not supported"	Third-party GGUF without effort levels	Use the staff-pick build, or the virtual-model wrapper on page 10.
Speed collapses mid-run on a Mac	Swapping: flash attention off or context too big	Flash Attention on, cache q8_0, smaller context. Check Activity Monitor → Memory → Swap Used.
Harness says connection refused	Server not running or wrong port	<code>lms status</code> , then <code>lms server start --port 1234</code> .
"Model not found"	Identifier mismatch	<code>lms ls</code> and copy the exact id; LM Studio loads it on demand.
Repeats itself with thinking off	Sampling left at thinking values	Temperature 0.7, top_p 0.8, presence penalty 1.5.

THE NUMBERS

What the whole stack costs

TOOL	JOB	START AT	UPGRADE WHEN
THE CORE			
Qwen 3.8-27B	The model	Free (Apache 2.0)	Never. Commercial use included.
LM Studio	Runs and serves it	Free, personal and work	Enterprise only for team SSO and model gating.
Claude Code	Harness A	Free CLI on the local route	A Claude plan when you want the frontier model for the hard 20%.
OpenCode	Harness B	Free, open source	Never.
Qwen Code	Harness C	Free, open source	Never on LM Studio. Its cloud free tier ended April 2026.
GitHub	Repos, PRs, tokens	Free, private repos included	Pro \$4/mo, rarely needed.
THE CONNECTORS			
GitHub MCP server	GitHub as tools	Free, hosted by GitHub	Never.
DuckDuckGo MCP	Web search	Free, no key	When you hit its rate limits daily.
Fetch MCP	Read any URL	Free	Never.
Playwright MCP	A real browser	Free	Never.
Tavily	Agent-grade search	Free, 1,000 credits/mo	Past 1,000 searches a month.
Brave Search API	Alternative search	\$5 credit/mo, attribution required	Metered after the credit, about \$3 to \$5 per 1,000 queries.
THE ONLY REAL COST: A MACHINE WITH ENOUGH MEMORY			
Mac, 32 GB+ unified memory	Laptop seat	Whatever you paid for it	36 GB M5 Max measured at 31 tokens/s with the drafter.
Used RTX 3090 (24 GB)	Desktop value seat	Second-hand market	41 to 63 tokens/s with the drafter.
RTX 5090 (32 GB)	Fastest single card	~\$4,600 to \$5,200 street, Sep 2026	MSRP is \$1,999. Wait for supply, or buy the 3090.

\$0/mo

Total software cost. Every line above the hardware band is free at the level this guide uses. The honest framing: you are converting a monthly AI subscription into a one-time hardware decision, and if you own a 24 GB machine already, that decision is made.

Set it up in this exact order

Do not download three quantisations and install every harness at once. Follow the sequence; each step proves the one before it.

1

Check your memory

About This Mac, or Task Manager → GPU.
Under 24 GB? Re-read page 6 before you download anything.

2

Install LM Studio, switch to Power User

Version 0.4.22 or newer. Open it once so the CLI can bootstrap.

3

Download the staff pick

qwen/qwen3.8-27b, Q4_K_M. On a Mac, grab the MLX 4-bit too and race them.

4

Load with the page 7 settings

Flash Attention on, cache q8_0, 32k context, Reasoning Medium. Save as default.

5

Turn on the drafter

Speculative Decoding → MTP. Time a 300-token answer before and after. Write the two numbers down.

6

Start the server

```
lms server start --port 1234, then curl  
localhost:1234/v1/models.
```

7

Install one harness

Claude Code or OpenCode. Run it on a throwaway repo. Ask it to describe the project.

8

Add git and the GitHub CLI

Fine-grained token, one repo. First pull request from your laptop model.

9

Add DuckDuckGo and Fetch

Ask it to find a library's current version with a source link.

10

Then, and only then, the GitHub MCP server

Read-only first. Widen it when a month of PRs has earned your trust.

The one-weekend challenge: Saturday is steps 1 to 7, and most of that is download time. Sunday is steps 8 to 10. By Sunday night you have a coding agent that reads your repos, searches the web and opens pull requests, running on a model that costs nothing per token and never sends your code anywhere.

FINAL WORD

The model is on your laptop now. *The meter is off.*

Eighteen months ago a model with these scores was a research-lab press release and a waiting list. Today it is a 17 GB download, an Apache licence, and a settings panel most people never open. The gap between "I heard local AI is slow" and "my laptop opened a pull request while I made coffee" is the five settings on page 8 and one weekend.

You do not need the 5090. You do not need the 128 GB Mac. You need the machine you have, the right file for its memory, reasoning set to medium, and one harness you actually use. Everything else in this guide is an upgrade you earn by shipping.

So pick a repo with a bug you have been ignoring. Point the agent at it tonight. Read the pull request it opens, and decide for yourself what the cloud is still worth to you.

KEEP GOING

More guides like this one, most of them free.

Guides, templates and courses from the studio: the 7 Tools to Get Paid stack, your own AI video editor, the Lead Engine and more.

b2blcc.com/downloads