Local Models × Claude Code

Architectures, Trade-offs & Possibilities for the Curious Engineer

Rob Brennan • Sploosh AI • github.com/TheRobBrennan

Where We Are in 2026

Anthropic API

Enterprise-grade but expensive ($5/$25 per M tokens for Opus)

Ollama + Local Models

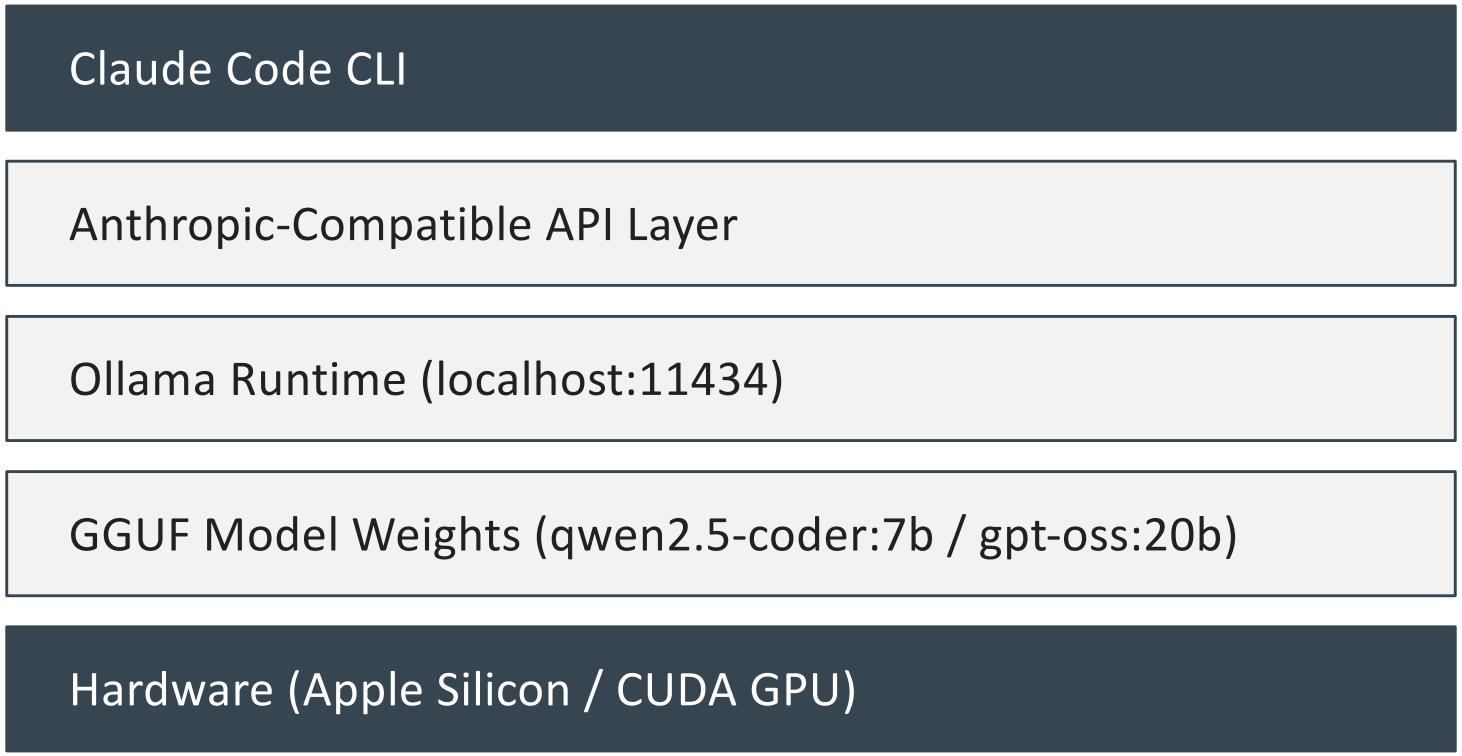
Free but trade-offs in capability

Hybrid Approaches

Best of both worlds?

The ollama launch command (v0.15+) provides native Claude Code compatibility — no manual env var configuration

Under the Hood



Ollama exposes an Anthropic-compatible API endpoint, allowing Claude Code to treat local models as drop-in replacements

Choosing Your Model

Model	Params	Size	Ctx	MCP	Speed	Quality
`qwen2.5-coder:7b`	7B	4.7GB	Std	Limited	Fast	Code focus
`gpt-oss:20b`	20B	11GB	Std	Full	Moderate	Broad
`qwen3-coder`	Large	~18GB	Std	TBD	Slower	Higher
`starcoder2:3b`	3B	1.7GB	Std	No	V. fast	Basic

Note: MCP tool execution requires models with robust function-calling capability

Model Context Protocol: The Extensibility Layer

fetch: URL content retrieval (no char limit by default)

simple-search: DuckDuckGo search via Python

Custom MCP servers: Build your own integrations

Only gpt-oss:20b reliably executes MCP tool calls among tested local models

Extend the Platform

Database MCP Server
→ Query your local/remote DBs from Claude Code

CI/CD MCP Server
→ Trigger builds, check pipeline status

Monitoring MCP Server
→ Pull metrics from Grafana/Prometheus

Internal API Server
→ Connect to your company's microservices

MCP is an open protocol — any service with an API can become a Claude Code tool

Mixing Local + Cloud Intelligently

Local-First, Cloud-Fallback

- Use local for day-to-day coding, switch to Opus for complex architecture decisions
- Estimated savings: 80-90%

Task-Based Routing

- Route simple tasks to local qwen, complex to cloud Sonnet/Opus
- Requires custom middleware

Tiered Development

- Junior devs: local models (cost-free exploration)
- Senior devs: cloud access for critical work
- CI/CD: batch API (50% savings)

Real-World Performance (M1 Max, 64GB)

Token Generation Speed

- qwen2.5-coder:7b: ~30-50 tok/s
- gpt-oss:20b: ~15-25 tok/s
- Cloud Sonnet 4.5: ~80-100 tok/s

Time to First Token

- Local: 200-500ms
- Cloud: 800-1500ms (network latency)

Context Window Handling

- Local: Limited by RAM
- Cloud: 200K+ tokens native

Local models win on latency; cloud wins on throughput and context

Tuning for Your Workflow

Context Length

`ollama launch` handles this, but manual config possible for edge cases

Environment Variables

ANTHROPIC_BASE_URL, ANTHROPIC_AUTH_TOKEN auto-set by ollama launch

Claude Settings

.claude/settings.json for project-specific config

IDE Integration

VS Code, Windsurf, Cursor — works standalone or integrated

Things to Try

1. **Fine-tune a model for your codebase style**
   Use Ollama's Modelfile to customize behavior
2. **Build a RAG pipeline with MCP**
   Index your docs, query through Claude Code
3. **Multi-model workflows**
   Use different models for different file types
4. **Local model evaluation framework**
   Benchmark models against YOUR specific tasks
5. **Team model server**
   Run Ollama on a shared GPU server, point all devs at it

Honest Trade-offs

Local Models Shine When	Cloud Models Win When
• Privacy is paramount	• Complex multi-file reasoning needed
• Cost elimination is the goal	• 200K+ context windows required
• Tasks are well-scoped (code gen, refactoring)	• Highest accuracy is critical
• Latency to first token matters	• Team standardization matters
• Offline development needed	• Function calling / tool use is essential

Fork It, Break It, Improve It

- npm scripts for everything — one-command setup
- Pre-configured MCP servers (web search + fetch)
- Example integrations (NHL API demo)
- GitHub Actions workflows included
- GPG-signed commits, conventional commits, PR templates

github.com/TheRobBrennan/how-to-setup-local-ollama-with-claude-code

Questions? Ideas? PRs Welcome.

rob@sploosh.ai

github.com/TheRobBrennan

Open source. Open to contributions. Open to possibilities.