Run Your Coding Agent on Local Models with PI Agent + Ollama

February 28, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** PI Agent is a free, open-source coding agent that runs on any model, including local ones via Ollama. Install PI, point it at your local Qwen 3.5 or Qwen3-Coder-Next, and you've got a private coding agent with zero API costs. It won't match Claude Code on frontier-model tasks, but a well-configured PI with a strong local model handles real coding work.

📚 **More on this topic:** Best Local Coding Models · llama.cpp vs Ollama vs vLLM · Best Local Models for OpenClaw · Local Alternatives to Claude Code

Most coding agents lock you into a single model provider. Claude Code requires Anthropic. GitHub Copilot requires OpenAI. You pay per token, your code goes through someone else's servers, and you have no control over what model runs underneath.

PI Agent is different. Built by Mario Zechner (the same developer behind OpenClaw), PI is a minimal, MIT-licensed terminal coding agent that works with any model from any provider, including local ones running on your own hardware via Ollama. No subscription, no API costs, no code leaving your machine.

This guide covers how to set it up, which local models work best for agentic coding, and what to expect from the experience.

What PI Agent actually is

PI is a terminal-based coding harness, not a full IDE plugin. You run it in your terminal, it reads and writes files, runs commands, and iterates on code. Think Claude Code's terminal mode, but open source and model-agnostic.

The design philosophy is aggressive minimalism:

Feature	PI Agent	Claude Code
System prompt	~200 tokens	~10,000 tokens
Default tools	4 (read, write, edit, bash)	10+ (read, write, edit, bash, glob, grep, web search, sub-agents…)

Feature	PI Agent	Claude Code
Model support	324 models across 20+ providers	Claude family only
Local model support	Ollama, vLLM, LM Studio, llama.cpp	None
Permission model	YOLO by default	5 safety modes, deny-first sandbox
Extension system	TypeScript in-process, 25+ event hooks	Shell-command hooks, 14 event types
Price	Free (MIT license)	$20–200/month or API keys
Sub-agents	Via extensions (spawn PI recursively)	Built-in (7 parallel)
MCP support	No (uses CLI tools instead)	Yes

The 200-token system prompt is a deliberate choice. Mario's argument: modern models have been RL-trained so heavily on coding agent tasks that they already know what to do. A 10,000-token prompt mostly tells the model things it already knows, while burning context that could go toward your actual codebase. Whether you agree with that philosophy depends on how much you trust your model.

The YOLO mode default is the other big philosophical difference. PI gives the agent full filesystem access and unrestricted command execution. No permission popups, no "allow this tool?" dialogs. The rationale: once an agent can write and execute code, any safety gate is theater. If you want permission gates, you build them yourself as an extension.

Setup: PI + Ollama in 5 minutes

Prerequisites:

- Node.js 14+ installed
- Ollama installed and running with at least one coding model pulled
- A GPU with enough VRAM for your chosen model (see model table below)

Step 1: Install PI

```
npm install -g @mariozechner/pi-coding-agent
```

Step 2: Pull a coding model in Ollama

```
# Pick one based on your hardware — see model table below
ollama pull qwen3.5:35b-a3b        # 16GB+ VRAM
ollama pull qwen3-coder-next        # 48GB+ VRAM/unified memory
ollama pull qwen2.5-coder:7b        # 8GB VRAM minimum
```

Step 3: Configure PI for Ollama

Create `~/.pi/agent/models.json`:

```
{
  "providers": {
    "ollama": {
      "baseUrl": "http://localhost:11434/v1",
      "api": "openai-completions",
      "apiKey": "ollama",
      "models": [
        { "id": "qwen3.5:35b-a3b", "name": "Qwen 3.5 35B-A3B", "contextWindow": 131072 },
        { "id": "qwen3-coder-next", "name": "Qwen3 Coder Next", "contextWindow": 256000 },
        { "id": "qwen2.5-coder:7b", "name": "Qwen 2.5 Coder 7B", "contextWindow": 32768 }
      ]
    }
  }
}
```

Create `~/.pi/agent/settings.json`:

```
{
  "defaultProvider": "ollama",
  "defaultModel": "qwen3.5:35b-a3b"
}
```

Step 4: Run it

```
cd your-project
pi
```

PI opens an interactive terminal session. Start typing what you want done. It'll read your files, write code, run commands, and iterate.

Switch models mid-session with `/model` if you want to try a different one without losing context.

Which local models work for agentic coding

Not every model that's "good at code" works well as a coding agent. Agent tasks require tool calling, multi-step reasoning, and the ability to recover from errors. A model that generates clean code in a single shot might fall apart when asked to read a file, edit a function, run tests, and fix failures in a loop.

Here's what actually works:

Model	Params (active)	VRAM (Q4)	Agent Quality	Speed (16GB GPU)	Best For
Qwen3-Coder-Next	80B (3B)	~47GB	Excellent	N/A (needs 48GB+)	Serious agent work on high-VRAM systems
Qwen 3.5 35B-A3B	35B (3B)	~8GB	Very good	~44 tok/s	Best balance for 16GB GPUs
Qwen3.5-27B dense	27B	~16GB	Very good	~15 tok/s	Dense model, needs 24GB for real context
Qwen 2.5 Coder 32B	32B	~20GB	Good	~10 tok/s (24GB)	Older but solid, needs 24GB
Qwen 2.5 Coder 7B	7B	~5GB	Decent	~90 tok/s	Quick tasks on 8GB cards
DeepSeek-Coder-V2 16B	16B (2.4B)	~10GB	Good	~50 tok/s	Decent MoE alternative

The pick: Qwen 3.5 35B-A3B

For most people running local coding agents, Qwen 3.5 35B-A3B is the model to use. It's a 35-billion parameter MoE model with only 3B active parameters per token. That means it fits on a 16GB GPU at Q4 with 100K context, generates tokens at ~44 tok/s, and still has the reasoning depth of a much larger model.

On SWE-bench Verified, the Qwen 3.5 series matches or exceeds models with 10x the active parameter count. The 27B dense variant hits 72.4, competitive with GPT-5 mini. The 35B MoE trades a small quality margin for much lower hardware requirements.

For agent loops specifically, speed matters as much as quality. A coding agent might make 20–50 tool calls per task, each burning hundreds of tokens. At 44 tok/s, Qwen 3.5 35B-A3B keeps the loop responsive. At 10 tok/s (Qwen 32B dense on 24GB), you're waiting 30+ seconds between each step. That's the difference between a tool you'll use daily and one that collects dust.

If you have more VRAM: Qwen3-Coder-Next

If you have 48GB+ of VRAM (Mac Studio with 96GB unified, dual RTX 3090s, or a workstation GPU), Qwen3-Coder-Next is the model to run. It scores 70%+ on SWE-bench Verified and 36.2 on Terminal-Bench 2.0, putting it in the same tier as frontier cloud models for agentic coding.

On an AMD Strix Halo with 64GB unified memory, community testing shows ~37 tok/s at 32K context. On dual 3090s with Q3 quantization, expect ~25 tok/s generation and ~1,000 tok/s prefill at 32K context. That's fast enough for productive agent work.

If you only have 8GB: Qwen 2.5 Coder 7B

It works. The model can follow basic tool-calling patterns, read files, and make edits. But it struggles with multi-step reasoning, frequently loses track of what it's doing in longer agent sessions, and generates more errors that need correction. Fine for "edit this function" or "write a test for this class." Not reliable for "refactor this module and update all the tests."

What the experience is actually like

Running PI with a local model is not the same as running Claude Code with Sonnet.

What works well:

- File reading, editing, and creation. The core loop is solid.

- Running tests and iterating on failures. This is where agents earn their keep, and a good local model handles it.
- Grep/search through a codebase to find relevant code before making changes.
- Simple refactors: rename a variable across files, extract a function, add error handling.
- Writing boilerplate: test files, config files, CI pipelines.

What gets shaky:

- Multi-file architectural changes. Frontier models handle these better because they can hold more context and reason about distant dependencies.
- Long sessions (50+ tool calls). Local models lose coherence faster than Claude Sonnet on extended tasks. PI's tree-based session structure helps — you can fork back to a good state when the model goes off track.
- Unfamiliar frameworks or libraries. Local models have less training data coverage for niche tools compared to frontier models trained on more data.

The harness compensates for model intelligence. Mario makes this point in his blog post, and he's right. PI's demos show Haiku (a bottom-tier cloud model) doing useful coding work, because the harness structure, hooks, and task management add determinism that the model itself lacks. The same applies to local models. A well-structured PI setup with AGENTS.md files, extension-based guardrails, and a strong local model handles most day-to-day coding tasks.

Basic customization

PI's extension system uses TypeScript. You don't need it for basic usage, but a few simple customizations make the local model experience better.

Project-level instructions (AGENTS.md)

Create an `AGENTS.md` file in your project root. PI automatically reads it as context:

```
# Project: my-app

## Stack
- TypeScript, Node.js 20, Express
- PostgreSQL with Drizzle ORM
- Jest for testing

## Rules
- Always run `npm test` after making changes
```

```
- Use existing patterns from src/ — don't invent new abstractions
- Import from @/lib, not relative paths
```

This helps more than anything else you can configure. It grounds the local model in your project's conventions without burning system prompt tokens.

Model switching for different tasks

You can switch models mid-session with `/model`. A practical workflow:

1. Start with Qwen 2.5 Coder 7B for quick lookups and small edits (fast, cheap on VRAM)
2. Switch to Qwen 3.5 35B-A3B for complex reasoning and multi-file changes
3. Switch back to 7B for running tests and fixing lint errors

PI tracks token usage and cost (even for local models, you can set notional costs to track context consumption). The `/model` command preserves your session history.

Slash commands via skills

PI supports reusable prompt templates as "skills." Create `~/.pi/agent/skills/review.md`:

```
description: Code review the staged changes
---
Run `git diff --cached` and review the changes.

Check for:
- Bugs or logic errors
- Missing error handling
- Naming that doesn't match the codebase
- Tests that should exist but don't

Be specific. Reference line numbers.
```

Then use `/review` in any PI session.

Limitations you should know about

PI makes tradeoffs. Some are deliberate design choices, some are gaps.

No MCP support. PI doesn't implement the Model Context Protocol. Mario's position is that MCP adds 7–9% context overhead for what's effectively a wrapper around CLI tools. PI's approach: install the CLI tool, give it a README, and the agent figures it out. This works fine in practice, but it means you can't reuse MCP servers you've already configured for other tools.

No built-in sub-agents. Claude Code can spin up 7 parallel sub-agents to handle tasks concurrently. PI doesn't have this out of the box. You can build it by spawning PI recursively via bash (`pi --print`), but it's manual work compared to Claude Code's `Task` tool.

TypeScript SDK for deep customization. If you want to write extensions that modify tool behavior, add custom UI elements, or intercept agent actions, you need TypeScript. This is a strength (in-process, 25+ event hooks, access to full session state) but also a barrier if TypeScript isn't your stack.

No IDE integration. PI is terminal-only. No VS Code panel, no JetBrains plugin. You work in your terminal alongside your editor. Some people prefer this; others find it disruptive.

No enterprise features. No SSO, no audit logs, no managed deployment. PI is for individual developers and small teams who want control over their tools.

Model quality ceiling. Even the best local models don't match Claude Sonnet on hard coding tasks. Qwen3-Coder-Next gets close (70%+ SWE-bench vs. Claude's ~72%), but Qwen 3.5 35B-A3B on a 16GB GPU is a step below that. You're trading some capability for privacy and zero cost. For most day-to-day coding tasks, the tradeoff is worth it. For hard debugging and architectural decisions, you might still want a frontier model.

PI vs. Claude Code vs. Aider: when to use each

Factor	PI + Local Model	Claude Code	Aider
Cost	$0 (local)	$20–200/mo or API	API costs only
Privacy	Full (nothing leaves your machine)	Code goes to Anthropic	Code goes to provider
Model flexibility	Any model, any provider	Claude only	Most models
Best model quality	Good (local) to Excellent (local + cloud mix)	Excellent	Excellent (uses frontier)
Setup complexity	Medium (Ollama + PI config)	Low (npm install, API key)	Low (pip install, API key)
	TypeScript, very flexible	Hooks + plugins	Limited

Factor	PI + Local Model	Claude Code	Aider
Extension system			
Git integration	Manual (via bash)	Built-in	Excellent (auto-commits)
IDE integration	None (terminal only)	VS Code, JetBrains	VS Code watch mode

Use PI + local models when: privacy matters, you want zero ongoing costs, you enjoy customizing your tools, or you're running coding agents on infrastructure you control.

Use Claude Code when: you need the best possible model quality, you want batteries-included with minimal setup, or you're working on hard problems where frontier model intelligence is the bottleneck.

Use Aider when: you want tight git integration with automatic commits, you prefer a Python ecosystem, or you want to mix cloud models without building your own configuration.

Getting started

1. Install Ollama and pull `qwen3.5:35b-a3b` (or whatever fits your GPU)
2. `npm install -g @mariozechner/pi-coding-agent`
3. Create `~/.pi/agent/models.json` and `settings.json` as shown above
4. Add an `AGENTS.md` to your project with stack and conventions
5. Run `pi` in your project directory

You'll know within 30 minutes whether the local model experience works for your workflow. If it does, you just saved yourself $20–200/month and gained full privacy over your codebase.

📚 **Related guides:** Best Local Coding Models 2026 · Qwen 3.5 Local Guide · llama.cpp vs Ollama vs vLLM · Local Alternatives to Claude Code · Best Local Models for OpenClaw

Get notified when we publish new guides.

Subscribe — free, no spam

Source: https://insiderllm.com/guides/pi-agent-local-models-ollama/

Free guides for running AI locally