OpenClaw Model Combinations: What to Pair for Each Task

March 5, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** One model can't do everything well in OpenClaw. The best setups pair models by task: Qwen 2.5 Coder 32B for autocomplete and code generation (~20GB Q4), Qwen 3.5 27B for chat, planning, and reasoning (~17GB Q4), and Qwen3-Coder-Next 80B-A3B for agentic coding if you have 48GB+ (~47GB Q4). On 24GB, run Qwen 2.5 Coder 32B as your primary and swap to Qwen 3.5 9B for chat. On 8GB, Qwen 3.5 9B does everything passably at 6.6GB.

📚 **More on this topic:** Best Local Models for OpenClaw · OpenClaw Setup Guide · Best Local Coding Models · VRAM Requirements

Most OpenClaw guides tell you to pick one model and use it for everything. I did that for months. It works, but you're settling for "okay at everything" when you could have "great at each thing."

OpenClaw skills can specify which model to use. A coding skill can route to a code-specialized model while a planning skill routes to a reasoning model. Different tasks have different requirements, and no single model is the best at all of them. Once I started pairing models by task type, the difference was obvious.

This guide covers which models to combine, why, and what fits on your hardware.

Why One Model Isn't Enough

Different OpenClaw tasks stress different capabilities:

Task Type	What It Needs	Best Model Type
Autocomplete / code gen	Fast, accurate code completion	Code-specialized (Qwen 2.5 Coder)
Chat / planning	Reasoning, instruction following, long context	General reasoning (Qwen 3.5, DeepSeek-R1)
Agentic coding	Multi-file edits, debugging, test writing	Large code agent (Qwen3-Coder-Next)

Task Type	What It Needs	Best Model Type
Tool use / function calling	Structured output, reliable JSON	Models with trained tool-call support

A code model writes better completions. A reasoning model plans better. Asking one model to do both is like using a screwdriver as a hammer.

The Models Worth Pairing

Qwen 2.5 Coder 32B — Best Autocomplete

Still the autocomplete king for 24GB cards. 92.7% on HumanEval, 128K context, and it's been the default in Continue and Copilot-style setups since late 2025.

~20GB at Q4_K_M. It handles tab completion, function generation, and boilerplate better than anything else local. It's terrible at planning, reasoning about architecture, or anything that isn't writing code. That's fine. That's not its job.

I keep this loaded as my default code model. Fast, accurate, predictable. For pure code generation on a 24GB card, nothing local beats it yet.

```
ollama pull qwen2.5-coder:32b
```

Qwen 3.5 27B — Best Chat and Reasoning

The 27B dense model scores 72.4 on SWE-bench and 85.5 on GPQA Diamond. 262K native context, built-in vision, and only ~17GB at Q4. That leaves 7GB of headroom on a 24GB card for context.

This is the model I reach for when I need to think through a problem, not just write code. Planning multi-step tasks, explaining why code is broken, architectural decisions. It's weaker than Qwen 2.5 Coder for raw completions, but stronger at everything around the code.

One gotcha: Ollama's tool-calling pipeline is broken for Qwen 3.5 as of March 2026. If your skills rely on function calling through Ollama, use Qwen 3 32B instead, or serve via llama.cpp or vLLM with the `qwen3_coder` tool-call parser.

```
ollama pull qwen3.5:27b
```

Qwen 3.5 9B — Best Budget All-Rounder

6.6GB on Ollama. Fits on 8GB cards with room for context. Beats models 3-4x its size on reasoning benchmarks, has 262K context, and handles vision natively.

I use this as my "quick question" model. Chat, simple planning, code review, anything where I don't want to wait for a bigger model to load. It won't handle complex agentic chains or multi-file refactors, that's what the 27B+ models are for, but for its size it's remarkably capable.

For low-VRAM setups, this is the model. See our Qwen 3.5 9B setup guide for the full walkthrough.

```
ollama pull qwen3.5:9b
```

Qwen3-Coder-Next 80B-A3B — Best Agentic Coding

This is the one that changed my mind about local agentic coding. An 80B MoE model that activates only 3B parameters per token, so it runs faster than the total parameter count suggests. 70.6% on SWE-bench Verified, which is real-world GitHub issue resolution, not toy benchmarks. 256K context.

~47GB at Q4_K_XL, ~37GB at Q3_K_XL. It handles multi-file edits, debugging across codebases, writing tests, resolving real issues. It can't run on a single 24GB card. You need dual GPUs or Apple Silicon with 48GB+ unified memory.

Performance: ~33 tok/s generation at 32K context on dual RTX 3090s. ~37 tok/s on 64GB unified memory systems. Prompt processing above 500 tok/s. And the context window barely costs VRAM, only ~7GB difference between 4K and 256K. That's the MoE architecture paying off.

```
# Needs 48GB+ VRAM or unified memory
ollama pull qwen3-coder-next
```

Devstral-Small-2-24B — Stable Agent Workhorse

If you need a model that just works for agent tasks without tool-calling bugs, this is it. Devstral-Small-2-24B has been running in production setups for weeks without failures. ~14GB at Q4_K_M, stable tool calling, 13+ tok/s on a 32GB Mac Studio.

It won't match Qwen 3.5 27B on raw benchmarks. But "never crashes" counts for a lot when you're running agents overnight. Reliable function calling and structured output, every time.

```
ollama pull devstral-small
```

Recommended Combinations by Use Case

Autocomplete + Chat (The Standard Pair)

Models: Qwen 2.5 Coder 32B + Qwen 3.5 9B

This is the combination I run daily on a single 24GB card. Qwen 2.5 Coder handles all code completions through Continue in VS Code. When I need to ask a question, plan an approach, or debug a logic error, I swap to Qwen 3.5 9B for chat.

Only one model loads at a time (Ollama handles the swapping), so you don't need VRAM for both simultaneously. The swap takes 3-5 seconds. Ollama's `OLLAMA_KEEP_ALIVE` setting controls how long each stays in memory.

Why not use Qwen 3.5 27B for chat? You can. But the 27B model takes longer to swap in, and for quick questions during coding, the 9B is fast and good enough. Save the 27B for dedicated planning sessions.

Planning + Execution (The Agent Pair)

Models: Qwen 3.5 27B (planner) + Qwen 2.5 Coder 32B (executor)

For agent workflows where OpenClaw needs to plan a multi-step task and then execute code changes, split the work. Route planning skills to Qwen 3.5 27B for its reasoning ability. Route code-writing skills to Qwen 2.5 Coder 32B for precise generation.

On 24GB, these swap rather than run simultaneously. On 48GB (dual GPUs), you could keep both loaded.

```
# In OpenClaw skill configuration:
# planning_skill → qwen3.5:27b
# coding_skill → qwen2.5-coder:32b
```

Full Agentic Coding (The Power Setup)

Models: Qwen3-Coder-Next 80B-A3B (primary) + Qwen 3.5 9B (fallback)

If you have 48GB+ VRAM (dual RTX 3090s, M4 Max 64GB+, or an M5 Pro/Max), Qwen3-Coder-Next is the model to run for serious agentic coding. It handles multi-file edits, test generation, and codebase-level reasoning at 70.6% SWE-bench. Keep Qwen 3.5 9B available as a fast fallback for simple chat queries — no need to burn Qwen3-Coder-Next's capacity on "what does this error mean?"

Low-VRAM Survival (8GB)

Model: Qwen 3.5 9B (everything)

On 8GB, you don't have room to pair models. Qwen 3.5 9B at 6.6GB handles chat, code, and simple agent tasks from one model. It punches well above its weight. For tasks it can't handle, route to a cloud API.

For harder agent work on 8GB hardware, see our upcoming OpenClaw on low-VRAM GPUs guide.

What to Run If You Have X VRAM

VRAM	Primary Model	Secondary (Swap)	Best For
8GB	Qwen 3.5 9B (6.6GB)	—	Chat, simple code, basic agents
12GB	Qwen 3.5 9B Q8 (13GB)	—	Better quality 9B, coding + chat
16GB	Devstral-Small-2-24B (14GB)	Qwen 3.5 9B (swap)	Agent tasks + quick chat
24GB	Qwen 2.5 Coder 32B (20GB)	Qwen 3.5 9B or 27B (swap)	Code completions + reasoning
32GB	Qwen 3.5 27B Q6 (24GB)	Qwen 2.5 Coder 32B (swap)	High-quality reasoning + code

VRAM	Primary Model	Secondary (Swap)	Best For
48GB	Qwen3-Coder-Next Q3 (37GB)	Qwen 3.5 9B (fast chat)	Agentic coding at SWE-bench 70.6%
64GB+	Qwen3-Coder-Next Q4 (47GB)	Qwen 3.5 27B (swap)	Full power agentic + planning

The “secondary” column means a model you swap to, not one you run simultaneously. Ollama handles model swapping automatically, though there’s a 3-10 second load delay depending on model size.

Context Length Matters More Than You Think

OpenClaw’s system prompt alone is roughly 17,000 tokens. Add conversation history, file contents, and tool outputs, and you’re burning through context fast. The minimum usable context for OpenClaw is 32K. For production agent work with sub-agents, 65K+ is recommended.

This changes the model math:

Model	Context at 24GB	Notes
Qwen 2.5 Coder 32B	~32K usable	20GB model + ~4GB KV cache at 32K
Qwen 3.5 27B	~65K+ usable	17GB model, 262K native, KV cache stays small thanks to Gated Deltanet
Qwen 3 32B	~16K usable	20GB model leaves less room

Qwen 3.5 27B has a real advantage here. Its Gated Deltanet architecture keeps KV cache growth much lower than standard attention, so you get more usable context per GB of VRAM. For OpenClaw specifically, where context is always under pressure, this matters.

Tool Calling: The March 2026 Gotcha

If your OpenClaw skills use function calling (and most agent workflows do), you need to know:

- **Qwen 3 32B:** Tool calling works in Ollama. Proven, reliable.

- **Qwen 3.5 27B:** Tool calling is broken in Ollama as of March 2026. Ollama routes it through a Hermes-style JSON pipeline instead of the Qwen3-Coder XML format the model expects. Works fine via llama.cpp or vLLM with the correct parser.
- **Devstral-Small-2-24B:** Tool calling works in Ollama. Community reports two weeks in production without a single failure.
- **Qwen3-Coder-Next:** Tool calling works. Built specifically for agentic workflows.

If you're using Ollama and need tool calling today, pair Qwen 3 32B or Devstral-Small with a code model. If you're willing to run llama.cpp or LM Studio, Qwen 3.5 27B is the stronger option.

Bottom Line

The single-model approach works for getting started. But once you've run OpenClaw for a week, you'll feel the limits. A code model that can't plan well. A reasoning model that writes sloppy completions. The fix is pairing.

On 24GB, my daily setup is Qwen 2.5 Coder 32B for completions plus Qwen 3.5 9B for chat. That covers 90% of what I need. For heavy agentic work, Qwen3-Coder-Next on 48GB+ is something else entirely. 70.6% SWE-bench from a local model was unthinkable a year ago.

Pick the combination that fits your VRAM, set up your skill routing, and stop asking one model to be good at everything.

Related Guides

- Best Local Models for OpenClaw
- OpenClaw Setup Guide
- Best Local Coding Models 2026
- OpenClaw on Low-VRAM GPUs
- VRAM Requirements for Every Local LLM
- Qwen 3.5 9B Setup Guide

Get notified when we publish new guides.

Subscribe — free, no spam

Source: https://insiderllm.com/guides/openclaw-best-model-combinations/