Best Models for Coding Locally in 2026

January 28, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** Qwen 2.5 Coder is the best local coding model family right now. On 8GB VRAM, the 7B variant scores 88.4% on HumanEval — better than models 3x its size. On 16GB, the 14B beats CodeStral-22B. On 24GB, the 32B matches GPT-4o. Pair any of them with Ollama + Continue in VS Code for a free, private Copilot replacement that works offline.

📚 **More on this topic:** VRAM Requirements · GPU Buying Guide · Ollama vs LM Studio

GitHub Copilot costs $10-19/month. ChatGPT Plus is $20. Claude Pro is $20. That's $120-240 per year for coding assistance that sends every line of your code to someone else's servers.

Local coding models flip that equation. You run them on your own hardware, your code never leaves your machine, and the total cost after setup is zero. The tradeoff used to be quality — local models couldn't compete with cloud. That changed in 2025. Open-source coding models now match GPT-4o on standard benchmarks, and they run on hardware most developers already own.

This guide covers which models to use at every VRAM tier, how they compare on real benchmarks, and exactly how to set them up in your editor.

Why Code Locally?

Four reasons developers are switching:

Your code stays private. Every prompt you send to Copilot or ChatGPT passes through corporate servers. If you're working on proprietary code, client projects, or anything sensitive, that's a risk. Local models process everything on your machine. Nothing leaves.

No recurring costs. $19/month for Copilot Business doesn't sound like much until you multiply it across a team or count years. Local models are free after the hardware investment — which you've probably already made if you have a GPU.

Works offline. Planes, coffee shops with bad WiFi, air-gapped environments, or just your ISP having a bad day. Local models don't care. No internet required.

No rate limits or surprise changes. No throttling during peak hours, no model swap without notice, no features removed from your tier. You control the model, the version, and the configuration.

What Makes a Good Coding Model

Not all LLMs are equal at code. The best coding models share four traits:

Code completion (FIM). Fill-in-the-middle support means the model can complete code given both the text before and after the cursor. This is what powers inline autocomplete in your editor. Not all models support FIM — coding-specific ones do.

Instruction following. "Refactor this function," "explain this regex," "write tests for this module." The model needs to follow natural-language instructions about code precisely, not just generate code from scratch.

Multi-language support. Most developers work across at least 2-3 languages. The model should handle Python, JavaScript/TypeScript, and your stack's other languages without falling apart.

Sufficient context window. Code context matters. A model that can only see 2K tokens is useless when your function depends on types defined 500 lines up. You want at least 8K, ideally 32K+.

The Benchmarks That Matter

Benchmark	What It Tests	Why It Matters
HumanEval	Generate correct Python functions from docstrings (164 tasks)	The standard for code generation quality
HumanEval+	Same tasks, 80x more test cases	Catches models that pass easy tests but fail edge cases
MBPP	974 programming problems	Broader than HumanEval, tests practical coding
MultiPL-E	HumanEval translated to 18+ languages	Shows if the model only knows Python or actually handles JS, Rust, etc.
LiveCodeBench	600+ real coding contest problems	Tests harder, more realistic tasks

HumanEval pass@1 is the most commonly reported number. Higher is better. For reference: GPT-4o scores ~90%, GPT-3.5 scored ~48% when it launched.

Best Models by VRAM Tier

8GB VRAM (RTX 4060, 3070, 3060 Ti)

This is the most common GPU tier for developers. The good news: the best 7B coding model now outperforms much larger models from a year ago.

Model	HumanEval	FIM	Context	VRAM (Q4)	Best For
Qwen 2.5 Coder 7B	88.4%	Yes	128K	~5 GB	Best overall. Start here.
DeepSeek Coder V2 Lite	81.1%	Yes	128K	~5 GB	Reasoning-heavy tasks
DeepSeek Coder 6.7B	~65%	Yes	16K	~4.5 GB	Lightweight, fast
CodeLlama 7B	~30%	Yes	16K	~4.5 GB	Legacy. Skip for new setups.

The winner: Qwen 2.5 Coder 7B. It's not close. An 88.4% HumanEval score at 7B parameters is remarkable — it beats CodeStral-22B (81.1%) and DeepSeek Coder 33B V1 (70%), models 3-5x its size. It supports FIM for autocomplete, has a 128K context window, and handles 92+ programming languages.

```
ollama pull qwen2.5-coder:7b
```

DeepSeek Coder V2 Lite is the runner-up. It's a 16B Mixture-of-Experts model that only activates 2.4B parameters per inference, making it fast and memory-efficient. Strong at reasoning-heavy coding tasks and math.

CodeLlama 7B is showing its age. At ~30% HumanEval, it's been lapped by models half its size from newer families. Only use it if you have a specific reason (e.g., Meta's Llama license requirements).

12-16GB VRAM (RTX 3060 12GB, 5060 Ti 16GB, 4060 Ti 16GB)

The mid-range tier opens up significantly better models. If you've got 16GB of VRAM, you can run 14B models at high quality or squeeze in a quantized 33B.

Model	HumanEval	FIM	Context	VRAM (Q4)	Best For
Qwen 2.5 Coder 14B	~89%	Yes	128K	~9 GB	Best at this tier

Model	HumanEval	FIM	Context	VRAM (Q4)	Best For
DeepSeek Coder 33B (Q3)	70%	Yes	16K	~16 GB	24GB model squeezed down
CodeLlama 13B	~36%	Yes	16K	~8.5 GB	Legacy. Outclassed.

The winner: Qwen 2.5 Coder 14B. It surpasses CodeStral-22B and DeepSeek Coder 33B on benchmarks despite being smaller. At Q4 quantization it needs only ~9GB, leaving room for long context on a 16GB card. State-of-the-art on over 10 code evaluation benchmarks.

```
ollama pull qwen2.5-coder:14b
```

DeepSeek Coder 33B at Q3 quantization technically fits in 16GB, but it's a tight squeeze with degraded quality. If you have exactly 16GB, the Qwen 14B at higher quantization (Q5 or Q6) will outperform it in practice.

24GB VRAM (RTX 3090, 4090)

This is where local coding gets seriously competitive with cloud models. A used RTX 3090 at $700-850 gives you access to models that match GPT-4o.

Model	HumanEval	FIM	Context	VRAM (Q4)	Best For
Qwen 2.5 Coder 32B	92.7%	Yes	128K	~20 GB	Best open-source coding model
DeepSeek Coder 33B	70% (78%*)	Yes	16K	~20 GB	Older but solid
CodeLlama 34B	53.7%	Yes	16K	~20 GB	Legacy. Outclassed.

*78% with CodeFuse fine-tuning

The winner: Qwen 2.5 Coder 32B. At 92.7% HumanEval, it matches GPT-4o. It scored 73.7 on the Aider benchmark (code repair), comparable to GPT-4o. It handles 40+ programming languages at high quality and has a 128K context window. At Q4_K_M (~20GB), it fits on a single 24GB card.

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

This is the model that made "local coding as good as Copilot" a real statement instead of wishful thinking. If you have a 3090 or 4090, this is what you should be running.

The Master Comparison

Every model side by side:

Model	Params	HumanEval	VRAM (Q4)	FIM	Context	License
Qwen 2.5 Coder 32B	32B	92.7%	~20 GB	Yes	128K	Apache 2.0
Qwen 2.5 Coder 14B	14B	~89%	~9 GB	Yes	128K	Apache 2.0
Qwen 2.5 Coder 7B	7B	88.4%	~5 GB	Yes	128K	Apache 2.0
DS Coder V2 Lite	16B (2.4B active)	81.1%	~5 GB	Yes	128K	MIT
DS Coder 33B (V1)	33B	70%	~20 GB	Yes	16K	Permissive
DS Coder 6.7B	6.7B	~65%	~4.5 GB	Yes	16K	Permissive
CodeLlama 34B	34B	53.7%	~20 GB	Yes	16K	Llama
CodeLlama 13B	13B	~36%	~8.5 GB	Yes	16K	Llama
CodeLlama 7B	7B	~30%	~4.5 GB	Yes	16K	Llama

The Qwen 2.5 Coder family dominates every tier. The 7B model beats all other sub-20B models. The 32B matches cloud-tier performance. All have Apache 2.0 licensing (commercial use allowed) and 128K context windows.

→ Check what fits your hardware with our Planning Tool.

Best Model by Language

All models above are multi-language, but some have particular strengths.

Language	Best Local Model	Notes
Python	Qwen 2.5 Coder (any size)	Best benchmarked language across all models
JavaScript/ TypeScript	Qwen 2.5 Coder 14B+	Strong JS/TS support; 7B handles it well too
Rust	Qwen 2.5 Coder 32B	Smaller models struggle with Rust's borrow checker; 32B handles it

Language	Best Local Model	Notes
Go	Qwen 2.5 Coder 14B+	Clean Go output from 14B up
C/C++	DeepSeek Coder 33B	Slightly better at low-level memory management patterns
Java	Qwen 2.5 Coder 14B+	Good boilerplate generation, understands frameworks
SQL	Qwen 2.5 Coder (any size)	82% on Spider benchmark — well ahead of competitors

The honest caveat: For Python and JavaScript, the 7B Qwen Coder is genuinely excellent. For Rust, C++, and other complex compiled languages, bigger models produce noticeably better results. If Rust is your primary language and you only have 8GB, expect some friction — the model will get syntax right but occasionally misunderstand lifetime annotations or trait bounds.

How to Set Up Local Coding in Your Editor

Option 1: VS Code + Ollama + Continue (Recommended)

This is the free, open-source Copilot replacement. Continue is a VS Code extension that connects to Ollama for both chat and autocomplete.

Step 1: Install Ollama

If you haven't already, follow our Ollama setup guide. One command on any OS.

Step 2: Pull your coding model

```
# Pick your tier:
ollama pull qwen2.5-coder:7b     # 8GB VRAM
ollama pull qwen2.5-coder:14b    # 16GB VRAM
ollama pull qwen2.5-coder:32b    # 24GB VRAM
```

Step 3: Install Continue extension

Open VS Code → Extensions (Ctrl+Shift+X) → Search "Continue" → Install.

Step 4: Configure Continue

Create or edit `~/.continue/config.yaml`:

```
name: Local Coding
version: 0.0.1
schema: v1
models:
  - uses: ollama/qwen2.5-coder-7b
```

For autocomplete (tab completion), Continue uses a separate, smaller model by default. You can also point it at your main coding model.

Step 5: Start coding

- **Chat:** Click the Continue icon in the sidebar, ask questions about your code
- **Autocomplete:** Start typing and suggestions appear inline
- **Edit:** Select code, press Ctrl+I, describe the change you want

Everything runs locally. No API keys, no accounts, no internet.

Option 2: LM Studio as Backend

If you prefer LM Studio's visual interface, you can use it as a backend for Continue too. Start LM Studio's local server, then point Continue at `http://localhost:1234/v1`.

This is useful if you like browsing and switching between models visually.

Option 3: Tabby (Self-Hosted Copilot)

Tabby is a self-hosted AI coding assistant with its own VS Code extension, JetBrains plugin, and Vim support. It's more opinionated than Continue — closer to a full Copilot replacement with built-in code indexing.

```
docker run -d --gpus all -p 8080:8080 \
  -v $HOME/.tabby:/data \
  registry.tabbyml.com/tabbyml/tabby serve \
  --model Qwen2.5-Coder-7B \
  --device cuda
```

Tabby works well for teams who want a shared local coding server. For solo developers, Continue + Ollama is simpler.

Option 4: Aider (Terminal)

Aider is a terminal-based coding assistant that edits files directly. Point it at your local Ollama instance:

```
pip install aider-chat
aider --model ollama/qwen2.5-coder:7b
```

It understands your git repo, makes edits across files, and creates commits. Best for developers who live in the terminal.

Practical Tips

FIM vs. Chat: Know the Difference

FIM (Fill-in-the-Middle) powers inline autocomplete — the cursor is in the middle of your code and the model predicts what goes there. This is what makes "tab complete" work. Qwen 2.5 Coder and DeepSeek Coder both support FIM.

Chat mode is for conversations: "explain this function," "refactor this class," "write tests." It's a different inference mode. Most editors let you use both simultaneously — FIM for autocomplete, chat for dialogue.

Keep a Small Model for Autocomplete

Autocomplete needs to be fast — under 200ms ideally. On 8GB, your main 7B coding model handles both chat and FIM fine. On 16-24GB, consider running a smaller model (Qwen 2.5 Coder 1.5B or 3B) for autocomplete and your bigger model for chat. This keeps tab-complete snappy while giving you full power for longer tasks.

Context Window vs. VRAM

Bigger context windows eat more VRAM. If you're working on a large codebase and want the model to "see" more files, you'll burn through your VRAM headroom fast. On 8GB, stick to 4-8K context for coding. On 16GB, you can push to 16K. On 24GB, 32K is comfortable.

For navigating large codebases, quantization at Q4_K_S instead of Q4_K_M saves a few hundred MB that can go toward context.

Close Your Browser

Same advice as for any 8GB VRAM workload: Chrome's hardware acceleration eats GPU memory. Close it or disable GPU acceleration when running local models. On 24GB this matters less, but on 8-16GB it can be the difference between smooth inference and OOM errors.

The Bottom Line

The Qwen 2.5 Coder family wins at every VRAM tier. Install the largest one your GPU can handle, connect it to VS Code with Continue, and you have a private, free, offline coding assistant that rivals Copilot on benchmarks.

Your setup in three commands:

```
# 1. Install Ollama (if you haven't)
curl -fsSL https://ollama.com/install.sh | sh

# 2. Pull your model (pick your VRAM tier)
ollama pull qwen2.5-coder:7b

# 3. Install Continue extension in VS Code, configure, and code
```

No subscriptions. No data leaving your machine. No rate limits. Just you, your code, and a model that actually knows what it's doing.

Related Guides

- How Much VRAM Do You Need for Local LLMs?
- GPU Buying Guide for Local AI
- Ollama vs LM Studio: Which Should You Use?

Sources: Qwen2.5-Coder Technical Report, DeepSeek Coder GitHub, Code Llama Paper, Continue.dev Ollama Guide, EvalPlus Leaderboard, Tabby

---

Free guides for running AI locally