Qwen 3.6 Complete Guide: 27B Dense, 35B-A3B MoE, and Which to Use

April 24, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** Two open models matter: Qwen3.6-27B dense (April 22, coding specialist, ~17GB at Q4) and Qwen3.6-35B-A3B MoE (mid-April, 3B active, ~22GB at Q4). Per Qwen's own benchmarks, the 27B scores 77.2 on SWE-bench Verified and beats the old 397B-A17B flagship on coding. The 35B-A3B hits 101 tok/s on an RTX 3090 at UD-Q4_K_XL per community benchmarks. Qwen3.6-Max-Preview is closed weights, cloud only. Skip it for local use. Pick 27B dense if you have 24GB VRAM and code for a living. Pick 35B-A3B if you have 16GB or less and want a general-purpose model that runs with RAM offload.

📚 **More on this topic:** Qwen 3.5 & 3.6 Cheat Sheet · Qwen 3.5 vs 3.6 GPU Fit Guide · llama.cpp vs Ollama vs vLLM · Qwen Mac MLX vs Ollama · VRAM Requirements

Qwen 3.6 shipped in three pieces over ten days. Qwen3.6-35B-A3B landed mid-April and took over the r/LocalLLaMA weekend threads. Qwen3.6-27B dropped on April 22 with a claim that a 27B dense model now beats the old 397B MoE on coding. Qwen3.6-Max-Preview arrived on April 20 as a cloud-only preview. Closed weights, no local option.

For a local-AI reader the question is simple. Which one do you run? The answer depends on one number: how much VRAM you have. That decides almost everything else.

Image: Qwen 3.6 variant comparison, 27B dense vs 35B-A3B MoE vs Max-Preview closed

Quick answer

- **Qwen3.6-27B dense.** Coding specialist. Apache 2.0, 262K context, ~17GB at Q4. Qwen claims 77.2 on SWE-bench Verified (per the Qwen team), which puts it in Sonnet 4.6 territory on agentic coding. Best for 24GB VRAM and up.
- **Qwen3.6-35B-A3B MoE.** 35B total, 3B active. Apache 2.0, 262K context, ~22GB at Q4. Community-tested 101 tok/s on an RTX 3090. Runs on 16GB VRAM with RAM offload. Best general-purpose option on modest hardware.
- **Qwen3.6-Max-Preview.** Closed weights, qwen.ai chat and API only. AA Intelligence Index score of 52 per Artificial Analysis, but no weights means no local run. Skip it for this guide.

For most readers the real decision is 27B dense vs 35B-A3B MoE. That's what the rest of this article is for.

What's new in Qwen 3.6

Three releases in the same ten-day window. Qwen shipped the 35B-A3B first in mid-April, then Max-Preview on April 20 as a closed-weights cloud preview, then the 27B dense on April 22. The 27B is the one Qwen labels "flagship." The 35B-A3B is the one that lit up local-AI threads before the 27B even existed.

Architecture-wise, Qwen 3.6 keeps the hybrid attention stack from 3.5. Three Gated DeltaNet layers for every one Gated Attention layer. The DeltaNet layers are linear attention, which is why these models don't hit a speed cliff at 64K+ context. Both open models are natively multimodal. A vision encoder is baked in, and both ship with mmproj files for image and video inference.

What changed since 3.5:

- **Agentic coding got the headline slot.** SWE-bench Verified moved 70.0 → 73.4 on the 35B-A3B and up to 77.2 on the new 27B dense. Terminal-Bench 2.0 went 40.5 → 51.5 on the MoE and 59.3 on the 27B dense. These are Qwen's own numbers from the model cards, so take them with salt. Multiple independent reports line up directionally.
- **Thinking preservation.** Both models were trained to keep `<think>` traces in context between turns. You can flip it on with `preserve_thinking: true` in chat template kwargs. Useful in agent loops where reasoning from turn 3 matters in turn 7.
- **Instruction-following nudged up.** Qwen's claim. Community mileage varies, and the MoE variant has picked up complaints here (more on that below).

The three variants at a glance

Model	Params	Architecture	License	Best for	Where to get it
3.6-27B	27B dense	Hybrid DeltaNet + Attention	Apache 2.0	coding, agentic work	Qwen, Unsloth GGUF
3.6-35B-A3B	35B total / 3B active	MoE (256 experts, 8+1 active)	Apache 2.0	general use, modest hardware	Qwen, Unsloth GGUF, bartowski
3.6-Max-Preview	undisclosed	closed	proprietary	cloud API only	qwen.ai chat and API

One note on Max-Preview. Per Artificial Analysis, it scores 52 on their Intelligence Index (rank #12 of 147), runs at 34.8 tok/s on their hosted benchmark, and costs $1.30/$7.80 per 1M tokens in/out. It's a closed Alibaba frontier play, not a local model. If you came here to pick something to run on your machine, it's not on the table.

Qwen 3.6-27B: The coding specialist

Qwen's framing is plain: "flagship-level agentic coding performance, surpassing the previous-generation open-source flagship." The previous flagship was the 397B-A17B MoE. So a 27B dense that fits in 17GB is claiming to beat a model that needed 214GB.

Per the Qwen 3.6-27B model card, the benchmark numbers:

Benchmark	Qwen3.6-27B
SWE-bench Verified	77.2
SWE-bench Pro	53.5
Terminal-Bench 2.0	59.3
LiveCodeBench v6	83.9
MMLU-Pro	86.2
GPQA Diamond	87.8
AIME 2026	94.1
HMMT Feb 2026	93.8

SWE-bench Verified at 77.2 is the number that made people pay attention. That puts Qwen3.6-27B in the same range as Sonnet 4.6 on agentic coding. An open-weight 27B model. On your hardware.

What the community actually saw. Simon Willison ran the Q4_K_M GGUF, a 16.8GB file, on llama-server at 65K context. He clocked 25.57 tok/s generating a pelican SVG and called it "an outstanding result for a 16.8GB local model." Nothing earth-shaking on the token rate, but the output quality is where people got excited. On an RTX 5090 with Q6_K, community posts report roughly 50 tok/s at 200K context. An RTX 5080 with 16GB gets about 6 tok/s at aggressive quant. Usable, not fast.

Honest caveats. The r/LocalLLaMA reaction split. One camp ran it for a week under Claude Code and OpenCode and reported it handles real agentic tasks that broke 3.5. Another camp pointed at the pelican-on-a-bicycle benchmark and asked whether Qwen trained on the test.

Hacker News commenters raised Goodhart's Law concerns and flagged tool-use failures where the model repeats failed actions without reading back context. Both camps are right about something. The 27B is strong at what it was trained for. That includes an awful lot of what people use coding models for. It is not a drop-in Claude replacement.

Sampling parameters matter. Qwen's card recommends temperature 0.6, top_p 0.95, top_k 20, presence_penalty 0.0 for "precise coding" in thinking mode. For everyday thinking-mode tasks, temperature 1.0, top_p 0.95, presence_penalty 0.0. Instruct (non-thinking) mode wants temperature 0.7, top_p 0.80, presence_penalty 1.5. Ignore these at your peril. Bad sampling turns good models into garbage generators.

Qwen 3.6-35B-A3B: The MoE workhorse

A3B is the magic. 35B total parameters but only 3B active per token. The model loads 35B of weights into memory and then routes each token through 8 of 256 experts plus 1 shared expert. You pay memory costs for 35B but compute costs for 3B. That asymmetry is why the 35B-A3B runs on hardware that can't touch a 32B dense model.

What it runs on. Per the Unsloth GGUF card, the quantization table:

Quant	Memory needed
UD-IQ1_M	10 GB
UD-Q3_K_M	16.6 GB
UD-Q4_K_M	22.1 GB
UD-Q4_K_XL	22.4 GB
Q8_0	36.9 GB
BF16	69.4 GB

A 24GB GPU runs UD-Q4_K_M clean. A 16GB GPU runs UD-Q3_K_M clean or UD-Q4_K_M with some KV cache offload. An 8GB GPU plus 64–96GB system RAM runs it through CPU/RAM offload. Slower, but usable for agent loops per Unsloth's docs.

Real throughput. Amine Raji's independent benchmark on an RTX 3090 at UD-Q4_K_XL: 101.7 tok/s short-prompt, 80.9 tok/s long-prompt. That's about 30% slower than Qwen 3.5-35B-A3B on the same card, a Gated DeltaNet implementation gap in current llama.cpp, not a model regression. Expect that gap to close over the next few llama.cpp releases. For iGPU/APU territory, community posts report Radeon 780M hitting roughly 250 tok/s prompt processing and

20 tok/s generation through Vulkan. Strix Halo (128GB unified) lands around 8–12 tok/s at bigger quants.

Where 35B-A3B disappoints. Community reports have been direct about this. On long multi-step agent tasks, the MoE variant has been described as "getting lost as the task requires more steps." Instruction-following under tight global rules has drifted in some harnesses. The pattern: tokens route to different experts, each expert handles its slice well, but cross-turn consistency suffers. None of this is published benchmark territory. It's week-one harness reports. But enough people have hit it to take seriously.

Speculative decoding. llama.cpp's speculative decoding support for MoE landed via PR #19493, but per community testing it doesn't net a speedup on A3B models. The routing overhead eats the draft-model win. Don't bother turning it on for 35B-A3B in llama.cpp today. The MTP path through vLLM and SGLang works better if you need speed.

MoE vs dense: how to choose

This is the real decision. Both models are Apache 2.0, both have 262K context, both have vision, both ship with thinking mode. The question is which architecture fits your workflow.

Pick 27B dense if:

- You have 24GB+ VRAM and don't need RAM offload
- You code primarily and want the best agentic-coding scores
- You've been burned by MoE drift on multi-step agent work
- You want strict instruction-following under heavy system prompts
- You're running a single task at a time and want peak quality per active parameter

Pick 35B-A3B MoE if:

- You have 16GB VRAM or less, or you want to run on a laptop with RAM offload
- You want raw speed on modest hardware. 3B-active generation is fast
- You're running general-purpose work and not exclusively coding agents
- You're OK with occasional MoE quirks on long task chains
- You want the broader knowledge footprint that 35B of stored weights gets you

The r/LocalLLaMA consensus, roughly: the dense vs MoE gap is shrinking, the MoE closed ground on 7 of 10 typical tasks between 3.5 and 3.6, but the 27B dense still wins most narrow well-defined tasks. Throw an ambiguous agent loop at both. Dense holds up better. Throw a chat workload at both. MoE wins on speed, ties on quality.

Image: MoE vs dense decision framework, VRAM, task type, speed trade-offs

Hardware quick guide

Skip this section if you've already read the GPU-fit cheat sheet. That article has the full quant-by-quant VRAM breakdown. Short version for the 3.6 lineup:

- **8GB VRAM + 32–96GB RAM.** 35B-A3B at UD-IQ1_M or UD-Q2 with CPU/RAM offload. Slow but real. Smaller Qwen 3.5 variants if you want better speed.
- **12GB VRAM.** 27B at Q3_K_M fits (~13.6GB, tight). 35B-A3B at UD-Q2_K_XL (~12.3GB). Neither is great. This is where you're fighting for headroom.
- **16GB VRAM.** 35B-A3B at UD-Q3_K_M (~16.6GB) is comfortable. 27B at IQ4_XS (~15.4GB) or Q4_K_S works. Sweet spot for 35B-A3B.
- **24GB VRAM (RTX 3090/4090/5090).** 27B at Q6_K (~22.5GB) or Q4/Q5 with big context. 35B-A3B at UD-Q4_K_M (~22.1GB). Both run well. 27B leaves more KV room.
- **32GB+ VRAM or Mac M3 Ultra / M4 Max 64GB+.** 27B at Q8_0 (~28.6GB), 35B-A3B at Q8_0 (~36.9GB). BF16 territory opens up above 70GB.
- **Mac M2/M3/M4 with 32–64GB unified memory.** MLX quants of both models run cleanly. Qwen on Mac MLX vs Ollama has the MLX-specific details.

The full GPU-by-GPU breakdown with tok/s numbers is in the GPU fit guide. Don't duplicate it in your head.

Backend compatibility: the Ollama bug

Your backend choice matters more than usual with 3.6. The depth comparison is in llama.cpp vs Ollama vs vLLM. The 3.6-specific status as of this writing:

- **llama.cpp.** The right choice for both models. Native GGUF support, handles the separate mmproj vision file, Gated DeltaNet works (just slower than it will eventually be). Recent llama.cpp builds required.
- **LM Studio.** Works. Wraps llama.cpp, handles the mmproj file for you, good Mac MLX path. This is what Simon Willison used for the 35B-A3B pelican test.
- **Ollama.** Broken for 3.6 GGUFs at the moment. Per Unsloth's 3.6 docs: "Currently no Qwen3.6 GGUF works in Ollama due to separate mmproj vision files." If Ollama is your whole stack, stay on Qwen 3.5-35B-A3B until this lands.
- **vLLM (≥0.19.0) and SGLang (≥0.5.10).** First-class support from Qwen's side. MTP speculative decoding works here. Best path for serving.

- **MLX on Mac.** Works. 3-bit through 8-bit quants available.

The CUDA 13.2 gotcha. Multiple community reports of low-bit 3.6 quants producing gibberish on CUDA 13.2. Pin CUDA 13.1 or a newer llama.cpp build. Amine Raji flagged it explicitly in his RTX 3090 benchmark: "Do NOT use CUDA 13.2 with Qwen3.6." If you see garbage tokens, check CUDA first.

Which one to download

Walk in with one question: how much VRAM do you have?

- **24GB.** Get the Unsloth Qwen3.6-27B-GGUF at Q6_K or Q8_0 if you code. Grab the 35B-A3B UD-Q4_K_M too if you want a general-purpose fallback.
- **16GB.** Get the Qwen3.6-35B-A3B-GGUF at UD-Q3_K_M. The 27B at IQ4_XS also fits but you'll have less KV headroom.
- **8–12GB plus 64GB+ system RAM.** 35B-A3B with aggressive quant and RAM offload. It's the only practical path.
- **Mac 32GB+ unified.** MLX builds of 35B-A3B. Q4 through Q6 depending on memory.
- **Anything less and you're working hard.** Smaller Qwen 3.5 variants still win here. Qwen 3.6 doesn't ship under 27B yet.

Don't chase Max-Preview. It's closed weights, cloud-only, and Alibaba is openly treating it as a different product line. If you wanted frontier closed models you'd already be on Claude or GPT. The local value is in the two Apache 2.0 models.

Bottom line

The 27B dense is a coding tool that happens to fit on a single 3090. The 35B-A3B is a general-purpose model that happens to run on a 16GB laptop GPU. Both are Apache 2.0. Both are natively multimodal. Both keep the hybrid attention that makes long context usable. If you have the VRAM, grab the 27B and put it to work on code. If you don't, grab the 35B-A3B and let MoE do what MoE does.

Ollama users, hold. llama.cpp users, run the CUDA check. Everyone else, pick a quant and see what your hardware does with it.

Source: https://insiderllm.com/guides/qwen-3-6-local-ai-guide/

Free guides for running AI locally