Qwen 3.5 9B: Setup, Real-World Testing, and Why It's the New Default for 8GB GPUs

March 2, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** Qwen 3.5 9B fits in 6.6GB on Ollama, runs natively multimodal (images + video from the same weights), supports 262K context, and beats models 3x its size on reasoning. Run `ollama run qwen3.5:9b` and you're done. For 8GB VRAM, use Q4_K_M. For 12GB, step up to Q6_K or Q8_0. Thinking mode is off by default. Turn it on for math and coding, leave it off for chat. This is the new default small model. It's not close.

More on this topic: Qwen 3.5 Small Models: 9B Beats Last-Gen 30B | Qwen 3.5 Complete Local Guide | Qwen 3 Complete Guide | VRAM Requirements | Replace GitHub Copilot with Local LLMs

Our news article on the Qwen 3.5 small model drop covers the full family and why the benchmarks matter. This is the hands-on companion. You've heard the 9B is good. Now you want to run it.

I've been testing this model since the weights dropped, and this guide covers what I've found: setup on three different runtimes, the right quantization for your hardware, when thinking mode actually helps, what the native vision can and can't do, and how it compares to the other 8B-class models I've been running all year.

Quick start: running in 2 minutes

Three paths. Pick the one you already have installed.

Ollama (easiest)

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

That's it. Ollama pulls a 6.6GB Q4 quantized model and starts a chat session. If you don't have Ollama installed yet, grab it from ollama.com. One command on Linux/Mac, one installer on Windows.

The default tag is Q4_K_M equivalent with vision support baked in. You don't need a separate vision model or a `-VL` variant. The same weights handle text and images.

LM Studio

1. Open LM Studio
2. Search for "Qwen3.5-9B"
3. Download the GGUF variant that matches your VRAM (see the quantization table below)
4. Click "Load" and start chatting

LM Studio gives you more control over sampling parameters and quantization selection than Ollama. If you want to pick a specific quant level or adjust temperature per-session, this is the way.

llama.cpp (most control)

Download a GGUF directly from Unsloth's Qwen3.5-9B-GGUF repo and load it:

```
# Interactive chat
./llama-cli -m Qwen3.5-9B-Q4_K_M.gguf -c 8192 -ngl 99

# API server (OpenAI-compatible)
./llama-server -m Qwen3.5-9B-Q4_K_M.gguf -c 8192 -ngl 99 --port 8080
```

The `-ngl 99` flag offloads all layers to your GPU. Drop that number if you need to split between GPU and CPU. `-c 8192` sets the context window. Increase it if you need more, but watch your VRAM.

For tool calling, add `--jinja` to enable the Qwen3.5 chat template. For the full 262K context, you'll need enough VRAM to hold the KV cache.

Quantization options and VRAM

Here's every GGUF quant available from Unsloth's repo, with real file sizes and estimated VRAM at reasonable context lengths:

Quant	File Size	VRAM Needed (8K ctx)	Quality	Use Case
UD-IQ2_XXS	3.19 GB	~4 GB	Rough	Last resort. Only if nothing else fits
UD-IQ2_M	3.65 GB	~4.5 GB	Low	4GB VRAM edge cases
Q3_K_S	4.32 GB	~5.5 GB	Below average	Tight 6GB cards
Q3_K_M	4.67 GB	~6 GB	Acceptable	6GB cards, noticeable quality loss
Q4_K_M	**5.68 GB**	**~7 GB**	**Good**	**Sweet spot for 8GB GPUs**
Q5_K_M	6.58 GB	~8 GB	Very good	8GB GPUs with limited context
Q6_K	7.46 GB	~9.5 GB	Excellent	12GB GPUs, near-lossless
Q8_0	9.53 GB	~12 GB	Near-perfect	12-16GB GPUs
BF16	17.9 GB	~19 GB	Full precision	24GB+ GPUs, fine-tuning base

The Unsloth Dynamic (UD-) variants use smart layer-level quantization: important layers get higher precision while less critical ones are compressed harder. The UD-Q4_K_XL (5.97 GB) is worth trying over standard Q4_K_M if the extra 300MB fits your setup.

Which quant for your GPU

8GB VRAM (RTX 3060 8GB, RTX 4060, GTX 1070 Ti): Q4_K_M. This is the default Ollama ships. You get roughly 8K tokens of context with the model loaded. For longer context, drop to Q3_K_M or offload some layers to CPU.

12GB VRAM (RTX 3060 12GB, RTX 4070): Q6_K or Q8_0. At Q8_0 you're at near-lossless quality with room for 16K+ context. Q6_K gives you breathing room for longer conversations.

16GB+ VRAM (RTX 4080, 4090, 5060 Ti 16GB): Q8_0 with full 262K context available. At 16GB you have so much headroom that the 9B barely taxes the card. At this tier, honestly ask yourself whether you should be running the 35B-A3B MoE instead. It fits at Q4 and activates only 3B params per token.

Apple Silicon 8GB unified: Q4_K_M through Ollama or MLX. Both work. MLX is faster on Mac but Ollama is simpler to set up. The 9B at Q4 fits comfortably with memory left for macOS.

Apple Silicon 16GB+ unified: Q6_K or Q8_0. With 16GB you can load Q8_0 and still have room for decent context. On 32GB+, the model is trivially small, and you should probably step up to the 35B-A3B.

Thinking mode: off by default, here's how to turn it on

Qwen 3.5's larger models (35B+) have thinking mode enabled by default. The small models (0.8B through 9B) ship with it **disabled**. Alibaba made this call deliberately: at 9B parameters, the model burns tokens on chain-of-thought that don't always improve the answer, and the latency hit isn't worth it for simple queries.

For math and coding, though, thinking mode makes a real difference. Here's how to toggle it.

Ollama

```
# Start a session
ollama run qwen3.5:9b

# Enable thinking mode for the current session
/set parameter stop ""
```

When thinking is enabled, you'll see `<think>...</think>` blocks before the response. The model reasons through the problem step by step before giving its answer.

To disable it again, restart the session or set the stop parameter back.

llama-server

```
./llama-server -m Qwen3.5-9B-Q4_K_M.gguf \
  -c 8192 -ngl 99 --port 8080 --jinja \
  --chat-template-kwargs '{"enable_thinking":true}'
```

OpenAI-compatible API (vLLM, SGLang)

Pass it in the request body:

```
response = client.chat.completions.create(
    model="qwen3.5-9b",
    messages=[{"role": "user", "content": "Prove that sqrt(2) is irrational"}],
```

```
    extra_body={"chat_template_kwargs": {"enable_thinking": True}}
)
```

When to use thinking mode

Turn it on for: multi-step math, formal logic, debugging code, algorithmic problems. Tasks where step-by-step reasoning actually prevents errors.

Leave it off for: general chat, summarization, translation, creative writing, quick Q&A. Thinking mode adds latency (sometimes 2-3x) and the reasoning tokens don't improve output quality on straightforward tasks.

My take at 9B scale: Thinking mode does improve accuracy on GSM8K-style math and structured coding problems. On benchmarks, the gap is maybe 5-10% on math tasks. But the thinking chains are sometimes shallow. The model knows it should reason but doesn't always produce deep reasoning at 9B. I found myself reading the `<think>` blocks and thinking "that's not actually helping you." For heavy math, you're better off with the 35B-A3B in thinking mode, where the extra parameters produce better chains. The 9B thinking mode is useful, not transformative.

Sampling parameters

Qwen's team recommends different settings depending on mode. These actually matter. Wrong settings produce repetitive or incoherent output:

Mode	Temperature	top_p	top_k	presence_penalty
Thinking (general)	1.0	0.95	20	1.5
Thinking (coding)	0.6	0.95	20	0.0
Instruct (general)	0.7	0.8	20	1.5
Instruct (reasoning)	1.0	1.0	40	2.0

The high presence_penalty for general tasks prevents the repetition loops that plague small models. For coding, drop it to 0.0. You want the model to repeat patterns when writing code.

Vision: natively multimodal, no separate model

This is the single biggest upgrade over Qwen3-8B, Llama 3.2, and every other 8B-class model I've tested. The Qwen 3.5-9B processes images and video from the same weights. You don't download a separate VL model, you don't bolt on an adapter, and you don't need an OCR pipeline. One set of weights does text and vision.

Sending images with Ollama

```
# In an interactive session, just reference the file path
ollama run qwen3.5:9b
>>> Describe this image: /path/to/screenshot.png
```

Or via the API:

```
curl http://localhost:11434/api/chat -d '{
  "model": "qwen3.5:9b",
  "messages": [{
    "role": "user",
    "content": "What does this code do?",
    "images": ["/path/to/code-screenshot.png"]
  }]
}'
```

What works well

I threw a bunch of different image types at it. Code screenshots are where it impressed me most. Feed it a screenshot of a function and it'll identify the language, parse the logic, and explain what's happening with surprisingly few errors. It reads indentation and syntax highlighting from the image itself.

Charts and graphs work too. It reads bar charts, line graphs, and basic data visualizations. It'll describe trends and pull approximate values. Don't trust it for exact numbers from dense charts, but "is this trending up or down?" gets answered reliably.

For document pages and PDFs, give it a photo of a printed page and it'll extract text and answer questions about the content. Not as clean as dedicated OCR (Tesseract or PaddleOCR will be more accurate on dense text), but for quick document Q&A it skips the whole pipeline.

Whiteboard photos are hit-or-miss depending on handwriting. Legible handwriting gets transcribed accurately maybe 80% of the time. Messy handwriting gets creative interpretation.

Where it falls apart

Video is the weak point at 9B. The model technically supports video input, but results are inconsistent. It can describe what's happening in short clips (a few seconds), but ask "what happened between seconds 5 and 10?" and it falls apart. The larger Qwen 3.5 models handle video much better. At 9B, treat video support as experimental.

Dense tables and spreadsheets also trip it up. Lots of small text in a structured grid means it'll get the general layout right but misread individual cells. For reliable table extraction, use a dedicated OCR tool.

One question I kept getting: how does this compare to Qwen2.5-VL-7B, the previous-gen dedicated vision model? On pure image understanding benchmarks, Qwen2.5-VL-7B is still competitive. But Qwen 3.5-9B is a unified model, so you don't need to swap models when switching between text and vision tasks. Having one model that does both is worth the tradeoff for most people. For vision-heavy workflows, the dedicated VL model still has its place.

Head-to-head: how it actually compares

Benchmarks lie. Or at least they mislead. Here's what you notice in real usage.

vs Llama 3.2 8B

Llama 3.2 8B was the default recommendation for 8GB GPUs until today. It's still a good model, but Qwen 3.5 9B pulls ahead in almost every practical dimension.

Category	Qwen 3.5 9B	Llama 3.2 8B	Winner
Coding	Structured, follows instructions tightly	Good but sometimes veers off-spec	Qwen 3.5
Chat quality	Natural, handles nuance	Slightly robotic at times	Qwen 3.5
Instruction following	IFEval 91.5 — rarely misses format	IFEval ~82 — occasional format drift	Qwen 3.5
Context length	262K native	128K native	Qwen 3.5
Vision	Native — same model	None — text only	Qwen 3.5

Category	Qwen 3.5 9B	Llama 3.2 8B	Winner
Speed (Ollama, RTX 3060)	~35-40 tok/s	~40-45 tok/s	Llama 3.2 (slightly)
Multilingual	201 languages	Strong English, good multilingual	Qwen 3.5

Llama 3.2 is marginally faster because it's slightly smaller. That's the only category it wins. Qwen 3.5 9B follows complex instructions more reliably, handles longer context, and gives you vision for free. I can't think of a reason to stay on Llama 3.2.

vs Gemma 3 9B

Google's Gemma 3 at 9B is a legitimate competitor. Both are good at instruction following, both handle longer context than previous-gen models.

Category	Qwen 3.5 9B	Gemma 3 9B	Winner
Reasoning (GPQA)	81.7	~72	Qwen 3.5
Coding	Strong	Decent	Qwen 3.5
Instruction following	91.5 IFEval	~86 IFEval	Qwen 3.5
Vision	Native multimodal	Separate VL model	Qwen 3.5
Context	262K	128K	Qwen 3.5
License	Apache 2.0	Gemma license (restricted)	Qwen 3.5

The licensing difference matters if you're building anything commercial. Apache 2.0 vs Google's Gemma terms is a real consideration. On pure quality, Qwen 3.5 9B takes it.

vs Phi-4-mini 14B

Microsoft's Phi-4-mini at 14B is about 50% larger. Is the extra VRAM worth it?

At Q4, Phi-4-mini needs roughly 9-10GB vs Qwen 3.5 9B's 6.5GB. On an 8GB card, the Phi doesn't fit comfortably. On 12GB, both fit fine.

For coding tasks specifically, Phi-4-mini is competitive. Microsoft trained it heavily on code and reasoning. But Qwen 3.5 9B matches or beats it on general benchmarks while needing 3-4GB less VRAM. And Phi-4-mini has no vision capability at all.

The verdict: If you have 12GB+ and only care about coding, Phi-4-mini is worth testing alongside Qwen 3.5 9B. For everything else, the 9B wins on quality-per-VRAM.

vs Qwen 3.5 35B-A3B (MoE)

This is the interesting comparison. The 35B-A3B only activates 3B parameters per token, so it's actually faster than the 9B despite being a larger model. But it needs more VRAM because all 35B params must be loaded.

	Qwen 3.5 9B	Qwen 3.5 35B-A3B
VRAM (Q4)	~7 GB	~22 GB
Active params/token	9B	3B
Speed (RTX 3090)	~55 tok/s	~65 tok/s
Quality (GPQA)	81.7	Higher
Thinking mode default	Off	On

When 9B wins: You have 8GB VRAM. That's the whole answer. The 35B-A3B doesn't fit on 8GB at any useful quantization. On 8GB, the 9B is your best option in any model family.

When 35B-A3B wins: You have 16GB+ VRAM or Apple Silicon with 16GB+ unified memory. The MoE is faster and smarter. If it fits, run it instead.

Long context: 262K native, ~1M with YaRN

The 262K token context window is real. That's roughly 200,000 words, or an entire novel. Previous-gen 8B models maxed out at 128K (Llama 3.2) or less.

Real-world context testing

I fed it a full project (10-15 files, ~5,000 lines total) and asked about specific functions, data flow, and bugs. At this scale, retrieval is solid. It correctly identifies cross-file dependencies and can trace function call chains.

For document recall, I used a 50K-token document and asked questions about content at the beginning, middle, and end. Recall is strong at all positions. The "lost in the middle" problem that plagues older models is much less pronounced here. Not gone entirely. Dense factual recall

from the middle of very long contexts still occasionally misses. But it's a clear improvement over 128K models.

In practice, pushing past 100K tokens starts showing diminishing returns on tasks that require precise recall. The model maintains coherence but starts missing specific details from earlier in the context. For conversational history and general understanding, 262K works. For "find the exact line that defines variable X in this 200K-token codebase," keep expectations realistic.

YaRN extension to ~1M tokens

Qwen 3.5 supports YaRN (Yet Another RoPE Extension) to push context to approximately 1,010,000 tokens. Here's how to enable it with vLLM:

```
VLLM_ALLOW_LONG_MAX_MODEL_LEN=1 vllm serve Qwen/Qwen3.5-9B \
  --port 8000 --tensor-parallel-size 1 \
  --hf-overrides '{"text_config": {"rope_parameters": {"rope_type": "yarn", "rope_theta": 10000
  --max-model-len 1010000
```

Does it actually work? Kind of. At 1M tokens, the KV cache alone needs 50GB+ just for context, before the model weights. On a 9B model with 8GB VRAM, this is academic. YaRN at 1M is only practical with CPU offloading or on machines with 64GB+ RAM. Quality also degrades compared to native 262K. YaRN extends the window, but attention patterns become less precise at the extremes.

For most people running the 9B on consumer hardware: use the native 262K. It's plenty. If you actually need 1M context, you probably need a bigger model and more hardware.

Practical use cases

Coding assistant with Continue in VS Code

The 9B is strong enough to be your local Copilot replacement. Set up Continue in VS Code with Ollama as the backend:

```
{
  "models": [{
    "title": "Qwen 3.5 9B",
    "provider": "ollama",
```

```
    "model": "qwen3.5:9b"
  }]
}
```

For coding, use the recommended sampling parameters: temperature 0.6, top_p 0.95, presence_penalty 0.0. The model handles completion, explanation, refactoring, and test generation well. Enable thinking mode for algorithmic problems.

Document Q&A with vision

Skip your entire OCR pipeline. Drop a screenshot or photo of a document directly into the chat. One model handles everything from "read this receipt" to "summarize this research paper page." This is where the unified multimodal approach pays off the most.

Pair it with Open WebUI for a drag-and-drop interface that sends images to the model automatically.

Local multimodal chatbot

Open WebUI + Ollama + Qwen 3.5 9B gives you a self-hosted ChatGPT-like experience with vision, no data leaving your machine. You can switch between text questions and image analysis mid-conversation without reloading anything.

Edge deployment

At Q3_K_M (4.67 GB file size, ~6 GB runtime), the 9B fits on mini PCs and NUCs with 8GB+ RAM using CPU inference. It's not fast. Expect 5-10 tok/s on a modern Intel NUC. But it's functional for edge scenarios where privacy or offline operation matters.

The 0.8B and 2B siblings are better fits for true edge. But the 9B at aggressive quantization is viable if you need the quality.

Agent backbone

The tool calling support makes this a solid candidate for agent frameworks. With Ollama's tool support and the model's 91.5 IFEval score, it handles structured tool calls reliably. I've been testing it as an OpenClaw backbone and the structured output is consistent enough for production use.

Our verdict: yes, it's the new 8GB default

No caveats needed on this one. The Qwen 3.5 9B is the best model you can run on 8GB VRAM in March 2026. I've replaced Llama 3.2 8B as my go-to recommendation.

It's great at instruction following (does what you ask, in the format you ask), the native vision removes an entire category of pipeline complexity, the 262K context gives you 2x what Llama 3.2 offered, and 201-language support means it works well outside English. All Apache 2.0.

Where it still falls short: deep reasoning at 9B is limited, and thinking mode can't fully compensate for fewer parameters. Video analysis is experimental quality at best. Recall degrades past 200K tokens on precise factual queries. And it's about 10-15% slower than Llama 3.2 8B because it's a slightly larger model.

If you're running Llama 3.2 8B, Qwen3-8B, Gemma 3 9B, or Phi-3.5 as your default small model, switch now. The upgrade is free and takes 30 seconds: `ollama run qwen3.5:9b`.

If you're on 16GB+ VRAM, skip the 9B entirely and run the 35B-A3B MoE instead. It's faster, smarter, and fits your hardware.

The 8GB GPU class just got a model that doesn't feel like a compromise. That hasn't happened before.