OPEN SOURCE · APRIL 2025

Meet Qwen3.6-35B-A3B
The Local AI That Changes Everything

Alibaba just released a model that runs on your laptop, thinks like an enterprise AI, and costs you nothing. Here's the complete guide to running it on any device.

35B total / 3B active params | 256K context window | Multimodal — vision + text | Thinking + non-thinking modes | Apache 2.0 — fully free | Runs on 13GB RAM (2-bit)

By Conneqt AI Team April 2025 15 min read

WHAT IS IT

Alibaba's Most Efficient Open Model Yet

Qwen3.6-35B-A3B is the latest model from Alibaba's Qwen research team, and it represents a meaningful leap forward for anyone running AI locally. It uses a **Mixture of Experts (MoE)** architecture — which means the model has 35 billion total parameters, but only 3 billion are activated at any given time during inference. Think of it like a team of 35 specialists: you only call in the 3 most relevant ones for each task.

The result is a model that performs like something much larger — on par with models 10 times its active size — while consuming the memory and compute of a 3B model. That's why the 2-bit quantized version can run on a standard 16GB laptop and still handle full codebase analysis, agentic tasks, vision, and multi-turn reasoning.

REAL-WORLD RESULT

Early users running the 2-bit version (13GB RAM) reported completing a full repository bug hunt — with evidence, reproduction steps, fixes, test cases, and a complete PR writeup — all from a single session on a consumer laptop.

The model is released under the **Apache 2.0 license**, meaning it's completely free to use commercially with no restrictions. You can deploy it in production, fine-tune it, and build products on top of it.

ARCHITECTURE

Why MoE Makes This Different

Traditional models (called "dense" models) activate all their parameters for every single token. A 35B dense model uses 35B parameters every time it processes a word. A 35B MoE model like Qwen3.6 uses only 3B parameters per token, routing each input to the most relevant subset of its expert layers.

Dense model (traditional)

All 35B parameters active at all times. High memory usage. Slower inference. Every parameter involved in every decision.

High resource cost

MoE model (Qwen3.6)

35B total, 3B active. Memory usage matches a 3B model. Fast inference. Routing selects the best experts for each task automatically.

10x efficiency

Qwen3.6 also builds on the Gated Delta Network architecture introduced in Qwen3.5, which combines sparse MoE with a more efficient attention mechanism. Combined with early-fusion training on text, code, and images simultaneously, this gives you **native multimodal understanding** — not a bolt-on vision module, but a model that was trained from scratch on visual data.

HARDWARE

What Device Do You Need?

The answer is probably whatever you already have. The model comes in multiple quantization levels — compressed versions that trade a small amount of quality for dramatically lower memory requirements. Here's what each version needs:

QUANTIZATION	RAM REQUIRED	QUALITY	BEST FOR
`UD-Q2_K_XL` — 2-bit	**~13 GB**		16GB laptops, older MacBooks, budget setups

QUANTIZATION	RAM REQUIRED	QUALITY	BEST FOR
`UD-Q3_K_XL` — 3-bit	**~17 GB**		32GB Mac, 16GB RAM + discrete GPU
`UD-Q4_K_XL` — 4-bit	**~22 GB**		M1/M2/M3 Mac 24GB+, NVIDIA 24GB VRAM

RULE OF THUMB

Your total available memory (RAM + VRAM combined) should ideally match or exceed the quantized model file size for the best performance. If it doesn't, inference still works — just slower due to disk offloading.

INSTALLATION

How to Run It — Every Method

Choose the method that fits your technical comfort level and use case:

Ollama | LM Studio | llama.cpp | Unsloth Studio | Cloud / API

Best for: Most users. Fastest setup. Works on macOS, Windows, Linux. No GPU required. Text-only (for image input, use llama.cpp).

01 Install Ollama from **ollama.com** — one installer for your OS. Opens as a background service.

02 Pull and run the model in your terminal:

TERMINAL Copy

```
ollama run qwen3.6:35b-a3b
```

Ollama automatically downloads the model (~13–22GB depending on device RAM) and starts an interactive session.

03 Switch between thinking and non-thinking modes at runtime:

TERMINAL Copy

```
# Enable deep reasoning (for coding, analysis)
/set think
```

```
# Disable for fast, direct responses (chat, writing)
/set nothink
```

04 Use the OpenAI-compatible API endpoint at `http://localhost:11434/v1` to connect n8n, Open WebUI, Continue.dev, or any other tool.

05 Example Python integration:

PYTHON Copy

```python
from ollama import chat

response = chat(
    model='qwen3.6:35b-a3b',
    messages=[{'role': 'user', 'content': 'Hello!'}]
)
print(response.message.content)
```

CORE FEATURE

Thinking Mode vs. Non-Thinking Mode

One of Qwen3.6's most powerful features is its ability to switch between two completely different reasoning behaviors — and you control this at runtime, with no model change needed.

Thinking mode

The model works through the problem internally before giving an answer — like showing its work. It uses a hidden chain-of-thought that you can optionally inspect.

Best for: debugging, math, code generation, complex analysis, multi-step reasoning, agentic tasks.

`/think` in chat, or `enable_thinking: true` in API

Non-thinking mode

Immediate, direct answers with no reasoning overhead. Significantly faster. Ideal for conversational flows and high-throughput automation.

Best for: chatbots, content generation, translation, Q&A, real-time tools.

`/nothink` in chat, or `enable_thinking: false` in API

USE CASES

What Can You Build With It?

🔍

Agentic Coding

Full codebase analysis, bug hunting, automated PR writeups, test generation. Proven on real repos at 2-bit.

👁

Vision Tasks

Analyze screenshots, UI images, documents, diagrams. Native multimodal — not a plugin.

⚙️

n8n Automation

Drop into any n8n OpenAI node via Ollama API. Local LLM that doesn't send your data anywhere.

💬

Private Chatbot

Run Open WebUI on top of Ollama for a full ChatGPT-like interface — completely offline.

🌍

Multilingual

201 languages including strong Arabic support. Ideal for GCC market products and Arabic NLP tasks.

🎛

Fine-tuning

Apache 2.0 means full training rights. Use Unsloth or transformers to adapt it to your domain.

INTEGRATION

Connecting to Your Existing Stack

Because Ollama and llama.cpp both expose an OpenAI-compatible API, Qwen3.6 drops into virtually any tool that supports OpenAI — no custom code needed.

n8n workflows

In any OpenAI node in n8n, set the credential base URL to `http://localhost:11434/v1` and the model to `qwen3.6:35b-a3b`. Your existing workflows work instantly with a local model — zero API costs, zero data leaving your machine.

Open WebUI

Open WebUI auto-detects Ollama on the same machine. Go to Settings → Connections → Ollama URL (`http://localhost:11434`). Your model appears in the model selector automatically.

Continue.dev / Cursor / Zed

Add Ollama as a provider in any of these IDE extensions. Use `qwen3.6:35b-a3b` as your autocomplete and chat model. The combination of speed (3B active params) and quality makes it one of the best local coding assistants available.

Claude Code / Clawbot / Codex / OpenCode

All of these agentic coding tools support Ollama as a backend. Point the base URL to `localhost:11434/v1`, set the model name, and run the same agentic workflows you'd run with a frontier model — locally, privately, for free.

OPTIMIZATION

Settings for Best Performance

For coding and agentic tasks

RECOMMENDED PARAMS

```
temperature: 0.7
top_p: 0.8
top_k: 20
thinking: ON
presence_penalty: 1.0 (reduces repetition)
```

For chat and content writing

RECOMMENDED PARAMS

```
temperature: 0.9
top_p: 0.95
top_k: 20
thinking: OFF
presence_penalty: 1.0
```

Context window

The default context in llama.cpp is 4096 tokens. For document analysis or long conversations, increase it:

BASH Copy

```
# 32K context (good balance of speed and length)
-c 32768
```

```
# 256K context (maximum, needs significant RAM)
-c 262144
```

IF THE MODEL KEEPS REPEATING

Add `--presence-penalty 1.5` to your llama.cpp command, or set `presence_penalty: 1.5` in your Ollama Modelfile. This is the most common issue with local MoE models and is easy to fix.

BOTTOM LINE

Why This Model Matters Right Now

The argument that "you need a powerful GPU to run serious AI locally" just expired. Qwen3.6-35B-A3B on 2-bit quantization is doing real agentic coding tasks on consumer hardware — not toy demos, but actual engineering work.

For builders in the GCC region specifically, this opens a different door: a model with **strong Arabic language support**, 201 total languages, vision capabilities, and a 256K context window — running privately on your own infrastructure with no API costs and no data leaving your hands.

The Apache 2.0 license means you can build, deploy, and sell products on top of it without restrictions. The MoE architecture means you get the quality of a 35B model at the inference cost of a 3B one. And the Unsloth Dynamic quantization means you can run it on hardware most people already have.

START HERE

If you have Ollama installed: `ollama run qwen3.6:35b-a3b` — that's it. The model downloads automatically, picks the right quantization for your hardware, and you're running enterprise-grade AI locally within minutes.