How to run Qwen 3.5 locally on a single GPU

Qwen 3.5 is Alibaba's latest Qwen model line, built upon the powerful performance of previous Qwen models in inference, programming, and multimodal tasks.

Qwen 3.5 is Alibaba's latest Qwen model line, built upon the powerful performance of previous Qwen models in inference, programming, and multimodal tasks.

Independent benchmark tests show that the Qwen 3.5-397B-A17B model scores highly in widely used benchmarks such as LiveCodeBench and AIME26, often outperforming leading models like GPT-5.2 and Claude Opus 4.5 in most evaluated categories, and delivering significantly higher throughput compared to previous Qwen generations.

Hardware and software requirements for Qwen 3.5

Before running Qwen 3.5 locally, you need to ensure your setup meets both hardware and software requirements for smooth inference. This guide will use an NVIDIA H200 GPU with 141GB of VRAM, combined with 240GB of system RAM, providing sufficient memory to efficiently run the MXFP4_MOE version of Qwen 3.5 with MoE offloading enabled.

In general, for best performance, your total VRAM + RAM capacity should approximately equal the size of the quantization model you are downloading. Otherwise, llama.cpp might transfer to an SSD , but the inference process will be slower.

Regarding software, you need to install the latest NVIDIA GPU driver, along with a recent version of the CUDA Toolkit, to ensure full compatibility with llama.cpp and CUDA-accelerated inference.

How to run Qwen 3.5 locally

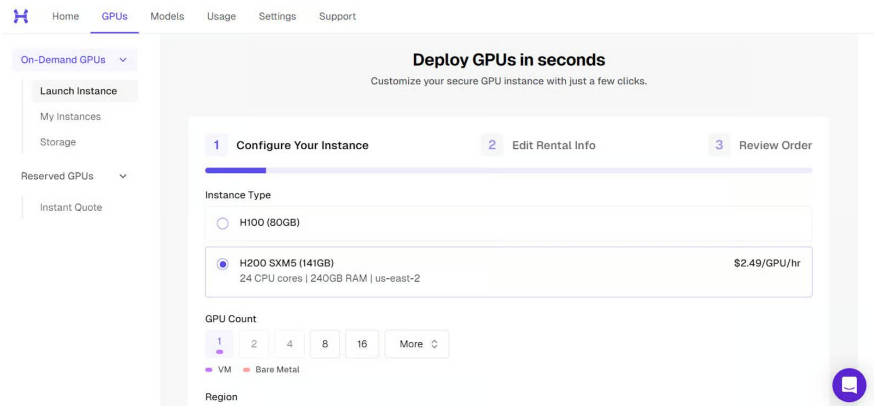
Now that you've met the prerequisites, let's look at a step-by-step guide on how to use Qwen 3.5 locally:

1. Set up the local environment.

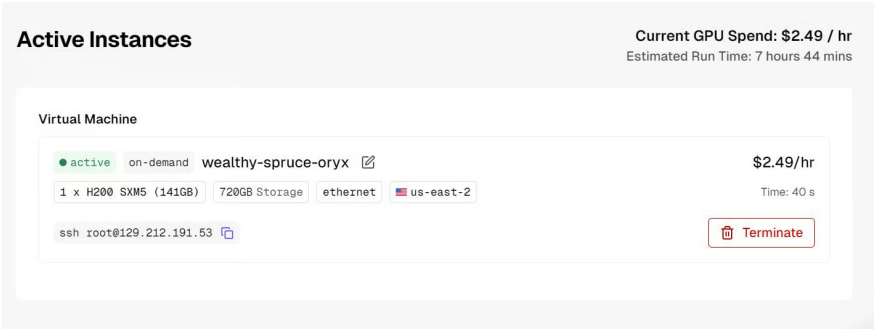
To run Qwen 3.5 locally, you need access to a computer with a powerful GPU . Since most laptops and desktops don't have enough VRAM or memory to handle models of this size, we'll be using a cloud-based GPU virtual machine.

This guide uses Hyperbolic to run the model privately. You can also use other providers such as RunPod, Vast.ai, or any GPU virtual machine platform you prefer. This article chooses Hyperbolic because it currently offers some of the most cost-effective GPU versions available.

Start by launching a new instance with a single H200 GPU.



After the machine boots up, you will see the public IP address and the SSH command needed to connect from your local terminal.

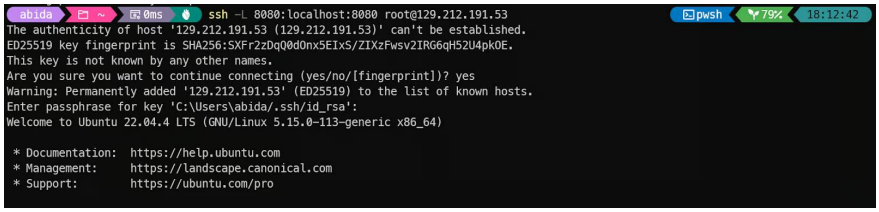


Before connecting, make sure you have set up local SSH and added your public SSH key when creating the virtual machine .

Once the instance is ready, connect to it via SSH with port forwarding. This is important because we want to access the local llama.cpp inference server through port 8080:

```
ssh -L 8080:localhost:8080 root@129.212.191.53
```

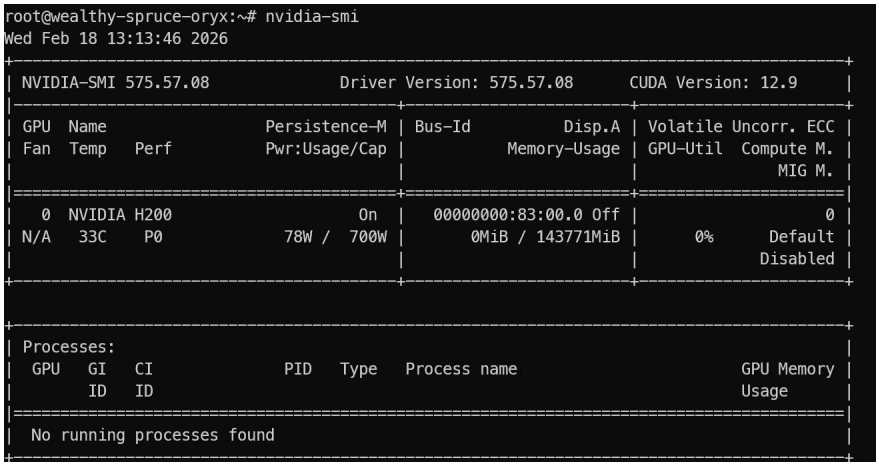
The first time you connect, type **yes** to confirm, then authenticate with your SSH key.



After logging in, verify that the GPU is correctly recognized:

```
nvidia-smi
```

You will see NVIDIA H200 listed in the results.



Finally, install the necessary Linux packages to download, compile, and run llama.cpp:

```
sudo apt update sudo apt install pciutils build-essential cmake curl libcurl4-ope
```

Once complete, your environment is ready to install llama.cpp and run Qwen 3.5 locally.

2. Install llama.cpp with CUDA support.

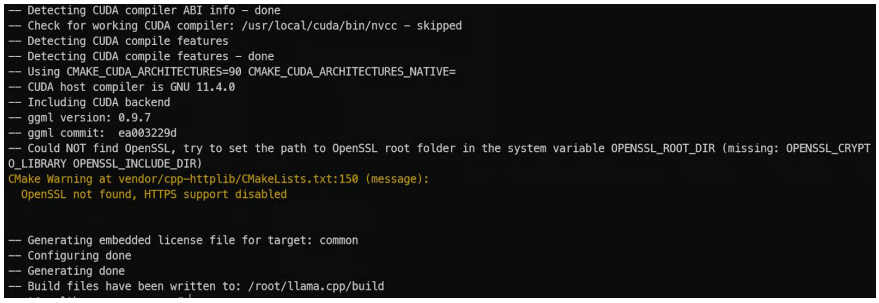
llama.cpp is an open-source C and C++ inference engine that allows you to run large language models locally with minimal setup, supporting both CPU and GPU acceleration.

First, copy the llama.cpp archive:

```
git clone https://github.com/ggml-org/llama.cpp
```

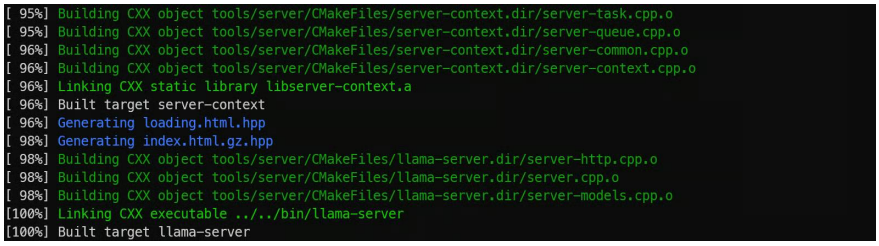
Next, we configure the build to support CUDA with CMake. We enable CUDA using -DGGML_CUDA=ON and set the CUDA architecture to 90a since we're using NVIDIA H200 (Hopper class). This helps the build generate GPU code optimized for Hopper features.

```
cmake llama.cpp -B llama.cpp/build  -DGGML_CUDA=ON  -DCMAKE_BUILD_TYPE=Release
```



Now compile the server binary file. llama-server is a built-in REST server that allows you to expose llama.cpp as an API endpoint:

```
cmake --build llama.cpp/build --config Release -j --clean-first --target llama-s
```



Finally, copy the compiled binary files to the home directory for easy execution:

```
cp llama.cpp/build/bin/llama-* llama.cpp
```

3. Download the Qwen 3.5 model.

Now that you've installed llama.cpp, the next step is to download the actual Qwen 3.5 model weights in GGUF format. These files are quite large, so using the Hugging Face CLI is the most reliable way to load them directly onto your GPU.

Python needs to be installed first because Hugging Face's download tools and validation utilities are distributed as Python packages. Although llama.cpp itself is written in C++, Python makes managing model downloads and delivery much easier.

Start by installing pip:

```
sudo apt install python3-pip
```

Next, install the Hugging Face Hub client along with performance-enhancing tools. hf_transfer and hf-xet significantly speed up downloads, which is crucial when downloading hundreds of gigabytes of model files.

```
pip -q install -U huggingface_hub hf-xet pip -q install -U hf_transfer
```

Now, let's download the Qwen 3.5 model from Hugging Face. In this tutorial, we'll only download the MXFP4_MOE variant, which is optimized for efficient MoE inference:

```
hf download unsloth/Qwen3.5-397B-A17B-GGUF  --local-dir models/Qwen3.5  --includ
```

Once the download is complete, the model files will be stored in models/Qwen 3.5, ready to be loaded into llama.cpp for local inference.

4. Launch the Qwen 3.5 model on a single GPU.

Now, we can launch Qwen 3.5 using llama-server. This provides us with an OpenAI-compatible endpoint API that can be called from local tools and applications.

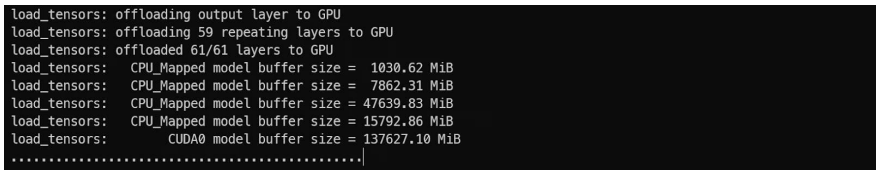
Optimize the server for a single-GPU setup by doing three main things. First, enable the **--fit** option so that llama.cpp automatically balances the model between the GPU's VRAM and system RAM, instead of reporting an error when the model doesn't fit in all the VRAM .

Secondly, we use a larger context window with --ctx-size 16384 so the server can handle longer prompts. Thirdly, we enable the --jinja option and pass --chat-template-kwargs to control chat formatting and disable thought mode for faster and more direct responses.

Run the server with the command:

```
./llama.cpp/llama-server  --model models/Qwen3.5/MXFP4_MOE/Qwen3.5-397B-A17B-MXFP
```

While the model is loading, you'll see it using both GPU VRAM and system memory, which is normal for a large MoE model.



Once the loading process is complete, the server will be accessible at:

1. 0.0.0.0:8080 on the virtual machine
2. http://127.0.0.1:8080 on your local machine after SSH port forwarding

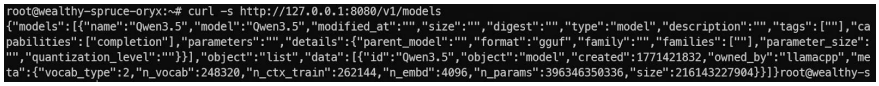
Let the server continue running. On your local computer, open a new terminal window and reconnect using SSH port forwarding:

```
ssh -L 8080:localhost:8080 root@129.212.191.53
```

Next, test the server by listing the available models:

```
curl -s http://127.0.0.1:8080/v1/models
```

If you see Qwen 3.5 in the response, your server is running correctly and you are ready to call it from the OpenAI SDK and your local applications.



5. Test the Qwen 3.5 model using the OpenAI SDK

Now that the Qwen 3.5 inference server is running, the next step is to verify that it works correctly with real-world client applications. One of the biggest advantages of llama.cpp is that llama-server provides an API compatible with OpenAI, meaning you can use the official OpenAI SDK without changing your code structure.

First, install the Python OpenAI package on your local computer (or inside a virtual machine if you prefer):

```
pip install openai
```

Now, let's run a simple test script. This script connects to your local forwarded endpoint at http://127.0.0.1:8080/v1 instead of OpenAI's cloud server.

```
python3 - 'PY' from openai import OpenAI client = OpenAI( base_url="http://127.0
```

There are a few important details to understand here:

1. base_url points to your local Qwen 3.5 server, not the OpenAI API.
2. The API key is still required by the SDK, but llama.cpp doesn't require authentication, so any placeholder value will work.
3. The model name="Qwen 3.5" matches the alias set when the server starts up.

If everything is configured correctly, you will receive quick and clear feedback from the model.

This confirms that:

1. The Qwen 3.5 model has been successfully loaded.
2. The llama.cpp server is running correctly.
3. Your SSH port forwarding is working.
4. The endpoint is fully compatible with OpenAI-style applications.

At this point, you can integrate Qwen 3.5 into any local tool, agent workflow, or application that already supports the OpenAI API format.

6. Develop a text-based user interface (TUI) for stock trading using Llama.cpp's WebUI.

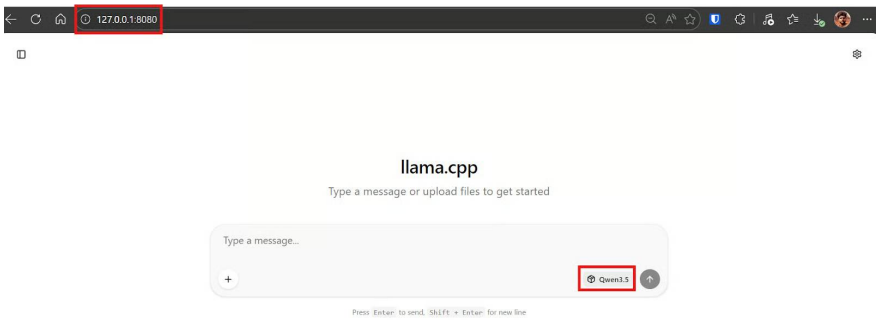
Llama.cpp includes a built-in WebUI, similar to ChatGPT , which you can use to chat directly with the model in your browser. This is useful for quick testing, iterative operations, and code generation without having to write any client scripts beforehand.

Because SSH port forwarding has been set up, you can open the WebUI on your local computer and it will function as if the server were running on your laptop.

By default, the WebUI is available at:

```
http://127.0.0.1:8080
```

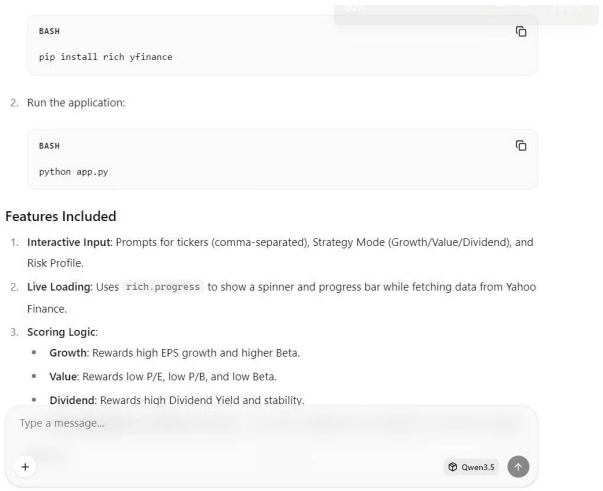
If this page loads, it confirms two things: your SSH tunnel is working correctly, and the Qwen 3.5 server is locally accessible while still running privately on a GPU virtual machine.



After you enter the WebUI, paste this prompt. The goal is for the model to generate both Python code and a brief user guide.

```
Xây d?ng m?t ?ng d?ng giao di?n ng??i dùng v?n b?n (TUI) ??n gi?n b?
ng Python "Stock Screener Trainer" ch?y b?ng `python app.py` s? d?ng th? vi
?n rich (không ph?i giao di?n web). ?ng d?ng này cho phép tôi nh?
p danh sách mã c? phi?u, ch?n ch? ?? (t?ng tr??ng/giá tr?/c? t?c) và m?c ??
 r?i ro (th?p/trung bình/cao), l?y các ch? s? c? b?n công khai cho m?i mã c
? phi?u t? m?t ngu?n mi?n phí, hi?n th? tr?ng thái load tr?c ti?p, sau ?ó t
?o m?t b?ng ??p và ph?n "Top 5 theo quy t?c ch?m ?i?m c?a tôi" v?i l?i c?
nh báo rõ ràng "ch? mang tính ch?t giáo d?c, không ph?i l?
i khuyên tài chính", và l?u toàn b? k?t qu? vào file `results.csv`.
```

Within seconds, Qwen 3.5 will generate an `app.py` file, usually a brief explanation of how to run it.



Now switch to your local terminal (laptop). Install the necessary libraries for the created application:

```
pip install rich yfinance
```

This operation installs:

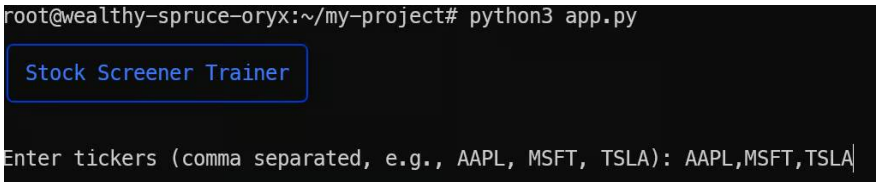
1. Rich for text-based user interface (TUI) layouts, tables, prompts, and progress indicators.
2. Use yfinance to get free, publicly available stock indices.

Create a file named app.py, paste the code generated by the model, and run it:

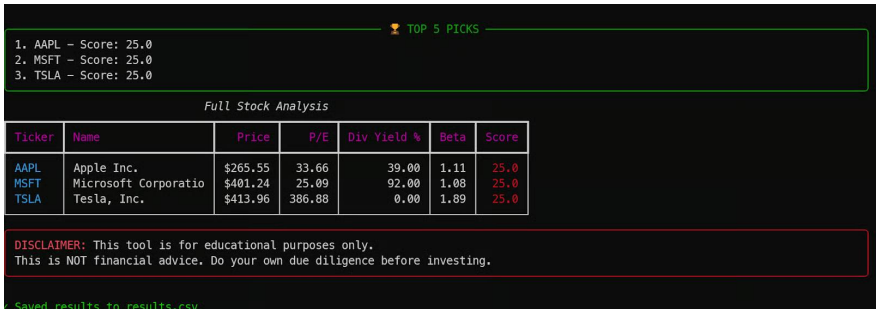
```
python3 app.py
```

After running the script, you will see the text-based user interface (TUI) launch correctly in your terminal. The application will prompt you to enter the stock ticker symbol you want to analyze, along with your preferred screening mode and risk level.

For example, the author of the article conducted experiments with three popular stocks.



After a short loading period, the tool will return a complete table of stock market indicators, highlighting the results based on scoring rules and saving everything to the results.csv file.



This is a great example of how Qwen 3.5 can create a fully functional application in a single go, using only a 4-bit quantized model endpoint and a simple prompt.