# Ollama

Run open models locally from the CLI or the REST API.



## 01 Install & start

```
curl -fsSL https://ollama.com/install.sh \
  | sh                      # macOS, Linux
docker run -d -v ollama:/root/.ollama \
  -p 11434:11434 --name ollama ollama/ollama
ollama --version
```

Windows: run OllamaSetup.exe from ollama.com/download. The API is at http://localhost:11434; ollama serve starts it.

## 02 Run a model

```
ollama run llama3.2       # chat
ollama run gemma3:27b     # pick a size tag
ollama run llama3.2 "Explain Docker"
echo "Summarize this" | ollama run llama3.2
ollama run llama3.2 "Summarize:" < doc.txt
ollama run gemma3 ./photo.png "Describe it"
```

run pulls the model if needed; no tag means :latest. In the chat, wrap multi-line input in """, set a system prompt with /set system and leave with /bye.

## 03 Manage models

```
ollama pull llama3.2          # download
ollama ls                     # local models
ollama show llama3.2          # details
ollama show --modelfile llama3.2
ollama ps                     # loaded now
ollama cp llama3.2 my-model   # alias
ollama rm my-model            # delete
ollama stop llama3.2          # unload
```

cp shares the layers, so an alias costs no disk. Models live in ~/.ollama/models (Linux service: /usr/share/ollama/.ollama/models).

## 04 Custom model (Modelfile)

```
FROM llama3.2
SYSTEM "You are a senior engineer."
PARAMETER num_ctx 8192
MESSAGE user "Which languages?"
MESSAGE assistant "Python, Go and Rust."
# ollama create my-coder -f ./Modelfile
```

FROM is required. TEMPLATE holds the Go prompt template ({{ .System }}, {{ .Prompt }}); LICENSE holds license text.

## 05 Model parameters

| | |
|---|---|
| temperature | Randomness; default 0.8 |
| num_ctx | Context window in tokens |
| num_predict | Max tokens; -1 = no limit |
| top_k / top_p | Sampling; default 40 / 0.9 |
| repeat_penalty | Above 1.0 curbs repeats |
| seed | Same seed and prompt, same text |

Also "options" per request or /set parameter in the chat. Default num_ctx: 4k under 24 GiB of VRAM, more above.

## 06 Generate a completion

```
curl localhost:11434/api/generate -d '{
  "model": "llama3.2",
  "prompt": "Write a haiku about code",
  "system": "You are a poet.",
  "options": {"temperature": 1.2},
  "stream": false
}'
```

Text is in "response"; for history use /api/chat.

## 07 Chat with history

```
curl localhost:11434/api/chat -d '{
  "model": "llama3.2",
  "messages": [
    {"role": "user", "content": "Hi"},
    {"role": "assistant", "content": "Hi!"},
    {"role": "user", "content": "Docker?"}]
}'
```

Resend the full history; reply is in message.content.

## 08 Request fields

| | |
|---|---|
| stream | false: one object, no chunks |
| options | Per-request parameters |
| format | "json" or a JSON schema |
| keep_alive | "10m", -1 keeps, 0 unloads |
| images / tools | Vision input, callable tools |

keep_alive defaults to 5m (OLLAMA_KEEP_ALIVE).

## 09 Structured output

```
curl localhost:11434/api/generate -d '{
  "model": "llama3.2",
  "prompt": "Tell me about Canada.",
  "format": {"type": "object",
    "properties": {
      "name": {"type": "string"},
      "population": {"type": "integer"}},
    "required": ["name", "population"]},
  "stream": false
}'
```

"format": "json" returns free-form JSON instead.

# Ollama

Vision, tools, embeddings, SDKs and server tuning.



## 10 Vision

```
curl localhost:11434/api/chat -d '{
  "model": "gemma3",
  "messages": [{"role": "user",
    "content": "Describe this image",
    "images": ["<base64 image data>"]}],
  "stream": false
}'
```

REST takes base64; the SDKs also take file paths.

## 11 Tool calling

```
from ollama import chat
def get_weather(city: str) -> str:
    """Get current weather for a city."""
    return f'Sunny, 22°C in {city}'
msgs = [{'role': 'user',
         'content': 'Weather in NYC?'}]
reply = chat(model='qwen3', messages=msgs,
             tools=[get_weather])
```

The SDK builds the schema from type hints and the docstring.

## 12 Return tool results

```
available = {'get_weather': get_weather}
msgs.append(reply.message)
for call in reply.message.tool_calls or []:
    fn = available[call.function.name]
    out = fn(**call.function.arguments)
    msgs.append({
        'role': 'tool', 'content': out,
        'tool_name': call.function.name})
final = chat(model='qwen3', messages=msgs)
```

Tool models: qwen3, llama3.2, command-r.

## 13 Embeddings

```
curl localhost:11434/api/embed -d '{
  "model": "nomic-embed-text",
  "input": ["First doc", "Second doc"]
}'
# {"embeddings": [[0.01, ...], [...]]}
```

input takes a string or a list. Prefer it to the older /api/embeddings ("prompt" in, "embedding" out). Also: all-minilm, mxbai-embed-large.

## 14 OpenAI-compatible API

```
from openai import OpenAI
client = OpenAI(
    base_url='http://localhost:11434/v1/',
    api_key='ollama')  # required, ignored
r = client.chat.completions.create(
    model='llama3.2', messages=msgs)
print(r.choices[0].message.content)
```

Also /v1/embeddings, /v1/models and /v1/responses. ollama cp llama3.2 gpt-3.5-turbo adds an OpenAI name.

## 15 Python SDK

```
# pip install ollama
from ollama import chat
msgs = [{'role': 'user', 'content': 'Hi!'}]
for part in chat(model='llama3.2',
                 messages=msgs,
                 stream=True):
    print(part.message.content, end='')
```

embed(), generate(), list(), pull() and ps() mirror the REST API. AsyncClient has the same methods with await.

## 16 JavaScript SDK

```
import ollama from 'ollama'
const stream = await ollama.chat({
  model: 'llama3.2',
  messages: [
    { role: 'user', content: 'Hi!' },
  ],
  stream: true,
})
for await (const part of stream) {
  process.stdout.write(part.message.content)
}
```

npm i ollama. Also embed(), list() and pull().

## 17 Server settings

```
export OLLAMA_HOST=0.0.0.0:11434  # network
export OLLAMA_MODELS=/data/models # storage
export OLLAMA_KEEP_ALIVE=10m      # idle
export OLLAMA_ORIGINS="*"         # CORS
OLLAMA_CONTEXT_LENGTH=8192 ollama serve
```

Defaults: 127.0.0.1:11434 and 5m. Linux service: set Environment="KEY=value" in systemctl edit ollama. macOS app: launchctl setenv.

## 18 GPU & performance

```
ollama ps        # PROCESSOR: GPU/CPU split
# parallel requests per model (default 1)
export OLLAMA_NUM_PARALLEL=4
# models loaded at once (default 3 per GPU)
export OLLAMA_MAX_LOADED_MODELS=2
```

GPUs are found automatically; Docker needs --gpus=all. OLLAMA_FLASH_ATTENTION=1 forces flash attention.