



Ollama is a framework for managing and serving both local and cloud-based language models, but local is where it really shines. Ollama pulls model weights, keeps an HTTP server on port 11434, and hands any client an OpenAI-shaped endpoint pointed at your own machine. Everything in this sheets assumes macOS on Apple silicon, where the GPU is reached through Metal and weights share one pool of unified memory with everything else you have open. Most of what follows works on Windows or Linux with negligible edits, if any. The question is never free VRAM, it is whether model plus context still fits.

ollama run

Pulls the model first if it is missing, then opens a chat. Bare ollama opens an interactive menu instead of running anything, within which Ollama configurations can be issued. You can launch with a prompt from command line as well.

```
ollama run gemma4 # chat, /bye to leave
ollama run gemma4 "summarize launchd briefly"

ollama # interactive menu
>>> /set parameter num_ctx 32768
```

pull · ls · show · cp · rm

Manage what is on disk. cp makes a second name for weights already pulled, which is how you satisfy tools that insist on an OpenAI model name.

```
ollama pull gemma4
ollama ls
ollama cp gemma4 gpt-3.5-turbo
ollama rm gpt-3.5-turbo
ollama show --modelfile gemma4
```

NAME	SIZE	MODIFIED
gemma4:latest	9.6 GB	2 days ago

ps · stop

Shows what is resident. PROCESSOR: under 100% GPU means part of the model spilled to CPU and generation slows to a crawl.

```
ollama ps
ollama stop gemma4
```

NAME	PROCESSOR	CONTEXT	UNTIL
gemma4	100% GPU	131072	4 minutes from now

serve

The menu bar app already runs a server, so this is for CLI-only installs or one you want in the foreground. --help lists every variable it reads.

```
ollama serve # foreground
curl http://localhost:11434 # Ollama is running
ollama serve --help # env var list
```

launchctl setenv

The macOS app is launched by the system, not your shell, so it never sees export lines in .zshrc. Set variables here, then restart the app (relaunch Ollama from the menu bar in macOS).

```
launchctl setenv OLLAMA_HOST "0.0.0.0:11434"
launchctl setenv OLLAMA_CONTEXT_LENGTH "64000"
launchctl getenv OLLAMA_HOST
```

~/ .ollama

Weights, logs and your public key all sit in the home directory. When the models outgrow the internal SSD, repoint them at an external volume.

```
~/ .ollama/models # weights, tens of GB
~/ .ollama/logs/server.log # server log
~/ .ollama/id_ed25519.pub # public key

launchctl setenv OLLAMA_MODELS /Volumes/AI/models
```

Context length

The default scales with memory: about 4K below 24 GB, 32K to 48 GB, 256K above. Agents and coding tools want 64K at minimum. Works from command line or within Ollama console.

```
OLLAMA_CONTEXT_LENGTH=64000 ollama serve
>>> /set parameter num_ctx 64000

ollama ps # CONTEXT shows what was allocated
```

Modelfile

A base model plus defaults, saved under a new name. It references the existing weights rather than copying them.

```
cat > Modelfile <<'EOF'
FROM gemma4
PARAMETER temperature 0.3
PARAMETER num_ctx 32768
SYSTEM ""Answer in one short paragraph.""
EOF
ollama create house-style -f Modelfile
```

/api/chat

The native endpoint. Streaming is the default, so pass stream: false for one JSON object. think and format control reasoning and schema.

```
curl http://localhost:11434/api/chat -d '{
  "model": "gemma4",
  "messages": [{ "role": "user",
                  "content": "why is the sky blue?" }],
  "stream": false
}'
```

/api/embed

Takes one string or a list. Vectors come back L2-normalized, so a plain dot product is already cosine similarity.

```
curl http://localhost:11434/api/embed -d '{
  "model": "embeddinggemma",
  "input": ["first sentence", "second sentence"]
}'

{"embeddings": [[0.013, -0.021, ...], [...]]}
```

OpenAI compatibility

Send an OpenAI client to /v1/ and it works: chat completions, embeddings, responses. The key is required by the client and ignored here.

```
from openai import OpenAI

client = OpenAI(
  base_url="http://localhost:11434/v1/",
  api_key="ollama" ) # required, ignored
```

ollama-python

The native client returns objects rather than dicts, and takes Ollama-only arguments like think that the OpenAI shape has no room for.

```
import ollama # pip install ollama

msgs = [{"role": "user", "content": "hello"}]
r = ollama.chat(model="gemma4", messages=msgs,
               think="low")
print(r.message.content)
```

Memory knobs

Models unload after five idle minutes. Parallel requests multiply the context allocation, so raise these only when the memory is actually there. Reminder that the following is macOS-specific.

```
launchctl setenv OLLAMA_KEEP_ALIVE "30m"
launchctl setenv OLLAMA_NUM_PARALLEL "2"
launchctl setenv OLLAMA_MAX_LOADED_MODELS "1"
launchctl setenv OLLAMA_KV_CACHE_TYPE "q8_0"
```

Tags · quantization

pull by default pulls the latest model. To better specify, the tag after the colon fixes parameter count and quantization. Budget ~0.6 GB per billion parameters at q4, twice that at q8_0, and remember that the KV cache still has to fit on top.

```
ollama pull gemma4:12b-q4_K_M
ollama show gemma4 # params, quant, context
```

parameters	12.2B
quantization	Q4_K_M
context length	131072

Structured output

Pass a JSON schema (model_json_schema()) as format and decoding is constrained to it, so the reply parses every time. The bare string "json" is looser: valid JSON, no promise about which keys arrive.

```
from pydantic import BaseModel

class Server(BaseModel):
    name: str
    port: int

r = ollama.chat(model="gemma4", messages=msgs,
               format=Server.model_json_schema())
Server.model_validate_json(r.message.content)

Server(name='ollama', port=11434)
```

MINING INSIGHTS IN DATA SCIENCE, MACHINE LEARNING, ANALYTICS, AND AI