河马的CC商业实战 · 会员专属资料 #5

零成本私有化大模型 · 从部署到生产

Ollama 本地部署手册

像管理容器一样管理大模型

三平台安装 · CLI 完全手册 · 模型选型决策树 · Modelfile 定制
API 与工具调用 · 性能优化 · 生态集成 · 企业级部署 · 十大常见问题排查

河马的CC商业实战 出品

2026年8月 · 会员专属版（内容基于2026年5月版）

会员专属资料 · 第 5 期

Ollama 本地部署手册

零成本私有化大模型从部署到生产实战指南

出品 河马的CC商业实战（知识星球）

版本 2026年8月 · 会员专属版（内容基于2026年5月版）

读者 想把大模型跑在自己机器上的星友——隐私、省钱、离线可用

> 本资料为「河马的CC商业实战」星球会员专属，每周/不定期更新。欢迎转发给需要的朋友——如果他们想持续收到这个系列，封底有入口。本地部署涉及具体版本号与命令，以你实际环境为准，思路不变。

河马卷首语

星友们，「会员专属资料」第 5 期来了。

这一期解决一个很多星友问过的问题：能不能不花 API 的钱、不把数据交给别人，就让大模型在自己电脑上跑起来？答案是 Ollama——一条命令装好，模型拉下来就能聊，断网也能用。这本手册从三个平台的安装讲起，CLI 命令、模型选型、API 调用、性能调优一路讲到企业级部署，最后还有十大常见问题的排查清单。

用法建议：第一章跟着装完跑通第一个模型，就有成就感了；第三章的模型选型决策树值得细看，别一上来就拉最大的模型；第九章排查清单建议收藏，遇到问题先去那里翻。

跑起来了记得来星球里晒一晒你的配置和速度。

—— 河马，2026年8月

目 录

FOREWORD

前言 为什么每个技术人都应该掌握 Ollama

数据隐私与成本控制是核心挑战

Ollama 的核心理念：像 Docker 管理容器一样管理大模型

Ollama 的五大核心价值

Ollama 能力架构全景图

CHAPTER ONE

第一章 安装与环境配置

1.1 macOS 安装

1.2 Linux 安装

1.3 Windows 安装

1.4 Docker 部署

1.5 安装后验证

CHAPTER TWO

第二章 核心 CLI 命令完全手册

2.1 基础命令

2.2 高级命令

CHAPTER THREE

第三章 模型生态完全指南

3.1 模型库与命名规则

3.2 量化级别完全手册

3.3 2026年5月推荐模型清单

3.4 模型选择决策树

CHAPTER FOUR

第四章 Modelfile 与模型定制

4.1 Modelfile 语法完整参考

4.2 参数调优指南

4.3 导入外部模型

4.4 Modelfile 模板合集

CHAPTER FIVE

第五章 API 完全手册

5.1 原生 API

5.2 OpenAI 兼容 API（重点）

5.3 结构化输出

5.4 工具调用（Function Calling）

5.5 多模态输入与流式输出

CHAPTER SIX

第六章 性能优化完全指南

6.1 GPU 加速

6.2 推理速度优化

6.3 内存与存储优化

6.4 性能基准测试

CHAPTER SEVEN

第七章 生态集成

7.1 桌面客户端

7.2 开发工具集成

7.3 自动化平台

7.4 知识库与 RAG

7.5 MCP 生态集成

CHAPTER EIGHT

第八章 企业级部署方案

8.1 单机多卡部署

8.2 负载均衡与高可用

8.3 监控与日志

8.4 安全加固

CHAPTER NINE

第九章 常见问题排查

1. GPU 不可用 / Ollama 仅使用 CPU

2. 内存不足 / 模型加载失败

3. 模型下载慢 / 国内镜像方案

4. 端口冲突

5. 模型加载失败 / GGUF 不兼容

6. 推理速度慢

7. API 调用超时

8. 中文输出乱码

9. Windows WSL2 无法连接宿主机 Ollama

10. 模型回答质量差 / 不符合预期

APPENDIX

附录

A. 各类硬件推荐配置表

A+. Apple Silicon 专用推荐配置

B. 所有环境变量速查表

C. Modelfile 模板合集

D. 学习资源推荐

D+. Ollama 版本升级指南

E. 免责声明

FOREWORD

前言 为什么每个技术人都应该掌握 Ollama

数据隐私与成本控制是核心挑战

Ollama 的核心理念：像 Docker 管理容器一样管理大模型

Ollama 的五大核心价值

Ollama 能力架构全景图

数据隐私与成本控制是核心挑战

2026年，大语言模型（LLM）已经深度融入软件开发的各个层面。然而，依赖云端 API 的方案正面临前所未有的挑战：企业核心数据频繁外泄、API 费用随调用量指数级增长、网络延迟导致交互体验不佳、以及服务中断带来的业务风险。在这样的背景下，**本地部署大模型**已经从极客玩具演变为严肃的生产级选项。

对于金融、医疗、法律等对数据敏感度极高的行业，将用户数据、商业文档甚至代码片段传输到第三方云端，往往意味着合规风险的急剧增加。欧盟 GDPR、中国《个人信息保护法》等法规的落地，让**数据不出域**成为企业刚性需求。与此同时，以 GPT-4o 为代表的顶级云端模型，其 API 成本对于高频调用场景（如内部知识库问答、代码补全、批量文档处理）而言，是一笔持续且可观的支出。以一支 20 人的开发团队为例，若每人每天调用 500 次代码补全，按月计算的费用足以采购一台配备 RTX 4090 的高性能工作站。

Ollama 的核心理念：像 Docker 管理容器一样管理大模型

Ollama 诞生于 2023 年，其设计哲学极其清晰：**将大模型变成和 Docker 镜像一样可拉取、可运行、可管理的标准化单元**。用户无需关心复杂的 Python 依赖、CUDA 版本、GGUF 转换或推理框架配置。一条 `ollama run llama3` 命令，即可在数分钟内启动一个本地可用的 Llama 3 实例。这种体验上的极致简化，使得 Ollama 迅速成为本地 LLM 领域的事实标准。

截至 2026年5月，Ollama 在 GitHub 上已获得超过 **165,000 Stars**，拥有超过 **52,000,000 次**月下载量和 700 余位贡献者。项目采用 MIT 许可证，完全开源免费，社区生态覆盖 LangChain、LlamaIndex、Open WebUI、Cherry Studio 等 40,000+ 集成项目。其版本迭代速度保持在每两周发布一个 minor 更新，v0.21.x 系列引入了 MLX logprobs 支持、更快的 fused sampling、Kimi CLI 集成以及 Gemma 4 的 Apple Silicon 原生优化，持续巩固其本地推理基础设施的领导地位。

Ollama 的底层基于 Georgi Gerganov 创建的 llama.cpp 项目，但做了大量易用性封装。它将 GGUF 格式的量化模型、推理后端（llama.cpp / MLX / CUDA / ROCm）、REST API 和多语言 SDK 统一到一个二进制文件中，用户感知到的只是一个命令行工具。这种设计使得 Ollama 既能满足开发者快速实验的需求，也能作为企业内网的 AI 推理层长期运行。

Ollama 的五大核心价值

1. **一条命令，模型就绪**：`ollama pull` 拉取模型，`ollama run` 即刻交互。无需编译，无需配置虚拟环境。
2. **原生 OpenAI 兼容 API**：暴露与 OpenAI 格式完全一致的 REST API，现有基于 `openai-python` 或 `openai-node` 的项目可零成本迁移。
3. **跨平台 GPU 加速**：自动识别 NVIDIA CUDA、AMD ROCm 和 Apple Metal，无需手动指定后端。
4. **多模态与嵌入原生支持**：除文本模型外，原生支持视觉语言模型（VLM）和文本嵌入模型，RAG 与图文理解一站解决。
5. **生产级扩展能力**：通过环境变量控制并发、多模型驻留、KV Cache 量化、结构化输出与 Function Calling，满足从个人实验到企业级部署的全场景需求。

Ollama 能力架构全景图



CHAPTER ONE

第一章　安装与环境配置

1.1 macOS 安装

1.2 Linux 安装

1.3 Windows 安装

1.4 Docker 部署

1.5 安装后验证

1.1 macOS 安装

本章覆盖 macOS、Linux、Windows 三大操作系统及 Docker 容器化部署方案。无论您是个人开发者、技术团队还是企业 IT 管理者，均能找到与自身环境匹配的安装路径。所有安装命令均经过 2026年5月最新版本验证，可直接复制执行。

官方 DMG 安装

访问 ollama.com/download，下载适用于 macOS 的安装包（支持 Apple Silicon 和 Intel）。双击 `.dmg` 文件，将 Ollama 拖拽至 Applications 文件夹即可完成安装。首次启动后，Ollama 菜单栏图标将出现，服务自动在后台运行。macOS 版本要求：macOS 12 (Monterey) 或更高，Apple Silicon 建议 macOS 14 (Sonoma) 以上以获得最佳 Metal 性能。

Homebrew 安装

对于偏好命令行且已安装 Homebrew 的开发者，使用 Homebrew 安装更为便捷，也便于后续升级管理：

```
brew install --cask ollama
```

安装完成后，运行以下命令验证：

```
ollama --version
# 预期输出：ollama version 0.21.3
```

Apple Silicon 优化专项

Ollama 在 Apple Silicon（M1/M2/M3/M4 系列）上通过 **MLX 后端** 实现极致优化，自动利用统一内存架构（Unified Memory）和 Metal GPU 核心。这意味着 M 系列芯片的 CPU、GPU 和 Neural Engine 共享同一物理内存池，模型数据无需在 CPU 内存和 GPU 显存之间复制，大幅降低了延迟和内存开销。无需额外配置，安装后即享硬件加速。

内存容量	推荐运行模型	典型量化	备注
8 GB	3B - 4B	Q4_K_M / Q8_0	M1/M2 基础款，适合轻量对话
16 GB	7B - 8B	Q4_K_M / Q5_K_M	M3/M4 Air 主力配置，代码生成可用
24 GB	13B - 14B	Q4_K_M / Q5_K_M	M3/M4 Pro 级别，可跑 Mistral Small
32 GB	14B - 27B	Q4_K_M	M4 Max 入门，多模态模型无压力
64 GB	32B - 70B	Q4_K_M / Q5_K_M	M4 Max 高配，可本地运行 Llama 3.3 70B
128 GB	70B - 110B+	Q4_K_M	M4 Ultra / M3 Ultra，企业级本地部署

> **M4 系列兼容性说明**：Ollama v0.21.x 已完全支持 M4、M4 Pro、M4 Max 及 M4 Ultra 芯片。M4 Ultra 的最高 512GB 统一内存配置意味着可以本地运行 400B 级量化模型，这在消费级硬件史上是前所未有的。对于 M4 Max 和 Ultra 用户，建议通过 `ollama show` 确认模型是否使用了 MLX 后端以获得最佳性能。
>
> **M4 系列兼容性**：Ollama v0.21.x 已完全支持 M4、M4 Pro、M4 Max 及 M4 Ultra 芯片，包括增强型 Neural Engine（38 TOPS）和更高内存带宽。实测 M4 Pro 运行 7B Q4_K_M 模型可达约 25 tok/s，M4 Max 运行 14B 模型可达约 25 tok/s，M4 Ultra 运行 70B 模型可达约 10-15 tok/s。

macOS 环境变量配置

macOS 上若通过图形界面启动 Ollama，环境变量需通过 `launchctl` 设置：

```
launchctl setenv OLLAMA_MODELS "/Volumes/ExternalSSD/ollama-models"
launchctl setenv OLLAMA_HOST "0.0.0.0:11434"
```

设置后需在菜单栏点击 Ollama 图标，选择 **Quit**，再重新启动应用生效。对于开发调试，建议通过终端直接启动 `ollama serve`，此时当前 shell 的环境变量会自动继承。

1.2 Linux 安装

官方一键脚本

Ollama 为 Linux 提供了自动安装脚本，支持 x86_64 和 ARM64 架构：

```
curl -fsSL https://ollama.com/install.sh | sh
```

该脚本将自动完成以下操作： - 下载与系统架构匹配的 Ollama 二进制文件 - 安装至 `/usr/local/bin/ollama` - 创建 `ollama` 用户与用户组 - 配置 systemd 服务以实现开机自启 - 打印后续使用提示

手动安装（含 systemd 服务文件）

若需自定义安装路径或审查安装过程，可手动执行：

```
# 1. 下载二进制文件
sudo curl -L https://ollama.com/download/ollama-linux-amd64 -o /usr/local/bin/ollama
sudo chmod +x /usr/local/bin/ollama

# 2. 创建用户（模型文件隔离）
sudo useradd -r -s /bin/false -U -m -d /usr/share/ollama ollama

# 3. 创建 systemd 服务文件
sudo tee /etc/systemd/system/ollama.service > /dev/null <<'EOF'
[Unit]
Description=Ollama Service
After=network-online.target

[Service]
ExecStart=/usr/local/bin/ollama serve
User=ollama
Group=ollama
Restart=always
RestartSec=3
Environment="PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"

[Install]
WantedBy=default.target
EOF

# 4. 启用并启动服务
sudo systemctl daemon-reload
```

```
sudo systemctl enable ollama
sudo systemctl start ollama
```

不同发行版注意事项

发行版	特殊说明
Ubuntu 22.04/24.04	官方脚本完全兼容，推荐首选
Debian 12	需确认 `curl` 和 `ca-certificates` 已安装：`sudo apt install curl ca-certificates`
CentOS / RHEL 9	需启用 EPEL 仓库：`sudo dnf install epel-release`
Fedora 40+	与 CentOS 类似，使用 `dnf` 安装依赖
Arch Linux	可通过 AUR 安装：`yay -S ollama` 或 `paru -S ollama`
openSUSE	使用官方脚本，Zypper 包管理器可额外安装 `lspci` 用于硬件检测

1.3 Windows 安装

官方安装包

访问 ollama.com/download 下载 `.exe` 安装程序。安装向导会自动配置环境变量并在后台启动服务。安装完成后，在 PowerShell 或 CMD 中运行：

```
ollama --version
```

> **2026 新特性**：Ollama 现已提供原生 Windows ARM64 构建，Surface Pro 等 ARM 设备不再依赖 x86 模拟，性能提升显著。对于 Windows 11 用户，推荐通过 winget 一键安装：`winget install Ollama.Ollama`。

WSL2 安装（推荐）

对于需要与 Linux 工具链深度集成的开发者，WSL2 是更优选择。WSL2 提供了完整的 Linux 内核兼容性，使得 Ollama 在 Windows 上能获得与原生 Linux 几乎一致的体验，同时支持与 Windows 宿主机共享 GPU 资源。

```
# 在 WSL2 Ubuntu 中执行
curl -fsSL https://ollama.com/install.sh | sh
sudo service ollama start
```

WSL2 的优势： - 与 Windows 宿主机共享 GPU（通过 CUDA on WSL） - 原生 Linux 路径和权限模型 - 与 Docker Desktop WSL 后端无缝协作

> **AMD GPU 在 WSL2 下的限制**：ROCm 对 WSL2 的支持尚处于实验阶段，AMD GPU 用户在 Windows 上建议直接使用原生安装或纯 CPU 模式。

Docker 安装

见 1.4 节 Docker 部署。

1.4 Docker 部署

Docker 是团队部署和生产环境的标准选择，确保环境一致性并简化迁移。

docker-compose.yml 完整示例

```
version: '3.8'

services:
  ollama:
    image: ollama/ollama:latest
    container_name: ollama-server
    ports:
      - "11434:11434"
    volumes:
      - ollama_data:/root/.ollama
    environment:
      - OLLAMA_NUM_PARALLEL=4
      - OLLAMA_MAX_LOADED_MODELS=2
      - OLLAMA_KEEP_ALIVE=5m
    restart: unless-stopped
    # GPU 配置见下方
    deploy:
      resources:
        reservations:
          devices:
            - driver: nvidia
              count: all
              capabilities: [gpu]

volumes:
  ollama_data:
```

GPU 直通配置（NVIDIA Container Toolkit）

在启用 GPU 直通前，需先在宿主机安装 NVIDIA Container Toolkit：

```
# Ubuntu / Debian
curl -fsSL https://nvidia.github.io/libnvidia-container/gpgkey \
  | sudo gpg --dearmor -o /usr/share/keyrings/nvidia-container-toolkit-keyring.gpg

curl -s -L https://nvidia.github.io/libnvidia-container/stable/deb/nvidia-container-toolkit.list \
  | sed 's#deb https://#deb [signed-by=/usr/share/keyrings/nvidia-container-toolkit-keyring.gpg] https://#g' \
  | sudo tee /etc/apt/sources.list.d/nvidia-container-toolkit.list

sudo apt-get update
sudo apt-get install -y nvidia-container-toolkit
sudo nvidia-ctk runtime configure --runtime=docker
sudo systemctl restart docker
```

验证 GPU 可用性：

```
docker run --rm --gpus all nvidia/cuda:12.0-base-ubuntu22.04 nvidia-smi
```

卷挂载和端口映射

配置项	说明	示例
端口映射	将容器 11434 端口暴露至宿主机	`-p 11434:11434`
命名卷	Docker 管理的持久化存储	`ollama_data:/root/.ollama`
绑定挂载	直接挂载宿主机目录	`/mnt/ollama:/root/.ollama`
多实例共享	多个 Ollama 容器共享模型目录	使用 NFS 或绑定挂载同一目录

生产环境建议使用绑定挂载至高速 SSD：

```
volumes:
  - /mnt/fast-ssd/ollama:/root/.ollama
```

1.5 安装后验证

版本检查

```
ollama --version
# 应输出类似：ollama version 0.21.3
```

服务状态检查

```
# Linux (systemd)
sudo systemctl status ollama

# macOS
launchctl list | grep ollama

# 通用 API 探测
curl http://localhost:11434/api/tags
```

快速测试

```
# 拉取并运行轻量模型进行端到端测试
ollama run phi4-mini "解释量子计算，限三句话"
```

若看到模型正常生成回复，说明安装成功。首次运行会自动下载模型，耗时取决于网络速度。

第一章总结：无论您使用哪种操作系统，Ollama 的安装过程都控制在 5 分钟以内。Docker 方案是团队协作和生产部署的首选，Apple Silicon 用户则享有当前消费级硬件中最优的本地推理体验。安装完成后，请务必通过 `ollama --version` 和 `ollama run phi4-mini "test"` 进行端到端验证，确保后续操作建立在可靠的基础之上。

CHAPTER TWO

第二章 核心 CLI 命令完全手册

2.1 基础命令

2.2 高级命令

2.1 基础命令

第二章覆盖从日常开发到高级定制的全部 CLI 操作。掌握这些命令后，您将能够高效管理模型生命周期、定制专属 AI 助手、以及精细化控制服务端行为。所有命令均基于 Ollama v0.21.3 验证。

`ollama run` — 运行模型（含交互模式命令）

`ollama run` 是最高频使用的命令，支持多种调用模式：

```
# 交互式对话模式
ollama run llama3.1:8b

# 单条指令非交互模式
ollama run qwen3.5 "请将以下英文翻译成中文：Hello world"

# 从管道输入
 echo "写一首关于编程的俳句" | ollama run gemma4:4b

# 保存输出到文件
ollama run llama3.1:8b "生成快速排序 Python 代码" > quicksort.py
```

交互模式内置命令：

命令	作用
`/set`	修改当前会话参数，如 `/set parameter temperature 0.5`
`/show`	显示当前模型信息，如 `/show info` 、 `/show modelfile`
`/clear`	清空当前对话上下文
`/bye`	退出交互模式
`/save <name>`	将当前对话保存为模型快照
`/load <name>`	加载已保存的对话快照
`/?` 或 `/help`	显示帮助信息

`ollama pull` — 从模型库拉取

```
# 拉取默认标签（通常是 Q4_K_M 量化）
ollama pull llama3.1:8b

# 拉取指定量化版本
ollama pull qwen3.5:14b-q5_K_M

# 从 HuggingFace 拉取 GGUF（见第三章）
ollama pull hf.co/bartowski/Llama-3.2-3B-Instruct-GGUF:Q8_0
```

`ollama list` — 列出本地模型

```
ollama list
# 输出示例：
# NAME                    ID           SIZE      MODIFIED
# llama3.1:8b             365c0bd3c000 4.7 GB    2 days ago
# qwen3.5:14b             aecbe5d06b0f 9.2 GB    1 week ago
# gemma4:e4b              8b2d22f2e1a4 15.1 GB   3 days ago
```

`ollama show` — 查看模型详情

`ollama show` 用于查看模型的元数据、Modelfile 源码、参数配置和系统提示词。在调试自定义模型或评估官方模型配置时非常有用。

```
# 查看完整信息
ollama show llama3.1:8b

# 仅查看 Modelfile 内容
ollama show llama3.1:8b --modelfile

# 仅查看参数
ollama show llama3.1:8b --parameters

# 仅查看系统提示词
ollama show llama3.1:8b --system

# 仅查看许可证
ollama show llama3.1:8b --license
```

通过 `--modelfile` 参数，可以学习官方模型的配置方式，进而基于其创建自己的定制版本。`--parameters` 则有助于排查生成行为异常，确认 temperature、context length 等关键参数的实际值。

`ollama ps` / `ollama stop` — 进程管理

```
# 查看已加载到内存中的模型
ollama ps
# 输出示例：
# NAME              ID           SIZE     PROCESSOR    UNTIL
# llama3.1:8b       365c0bd3c000 4.7 GB   100% GPU     4 minutes from now

# 停止指定模型（释放显存）
ollama stop llama3.1:8b

# 停止所有模型
ollama stop --all
```

`ollama rm` / `ollama cp` — 模型管理

```
# 删除本地模型（释放磁盘空间）
ollama rm codellama:7b

# 复制模型（创建别名或备份）
ollama cp llama3.1:8b mycompany/llama3.1-custom:8b

# 清理所有未使用的模型层（类似 docker prune）
ollama prune
```

2.2 高级命令

`ollama create` — 从 Modelfile 创建自定义模型

`ollama create` 是模型定制的核心入口。假设有以下 `Modelfile`：

```
FROM llama3.1:8b

SYSTEM """你是一个专业的 Python 开发助手。遵循 PEP 8 规范，优先使用标准库，
代码注释使用中文，并在必要时提供单元测试示例。"""

PARAMETER temperature 0.2
PARAMETER top_p 0.9
PARAMETER num_ctx 8192

MESSAGE user "写一段读取 CSV 的代码"
MESSAGE assistant "```python\nimport csv...\n```"
```

创建模型：

```
ollama create python-assistant -f Modelfile
ollama run python-assistant
```

`ollama serve` — 启动服务的全部环境变量选项

`ollama serve` 启动 API 服务器（默认端口 11434）。以下环境变量控制服务器行为：

```
# 网络绑定
export OLLAMA_HOST=0.0.0.0:11434        # 监听所有接口（生产环境需配合防火墙）

# 模型存储
export OLLAMA_MODELS=/mnt/fast-ssd/ollama-models

# 并发与驻留
export OLLAMA_NUM_PARALLEL=4            # 单模型并发请求槽位数
export OLLAMA_MAX_LOADED_MODELS=2       # 内存中同时驻留的最大模型数
export OLLAMA_KEEP_ALIVE=5m             # 模型空闲驻留时长（0=立即卸载，-1=永久驻留）

# GPU 与性能
export OLLAMA_NUM_GPU=99                # 加载到 GPU 的层数（99=全部）
export OLLAMA_FLASH_ATTENTION=1         # 启用 Flash Attention（推荐）
export OLLAMA_KV_CACHE_TYPE=q8_0        # KV Cache 量化：f16 / q8_0 / q4_0
```

```
# 调试
export OLLAMA_DEBUG=1                    # 启用调试日志
```

第二章总结：`ollama run`、`pull`、`list` 等基础命令构成了日常使用的 80%。当需要定制模型行为时，`ollama create` 配合 Modelfile 提供了无限可能。而 `ollama serve` 和环境变量的组合则是将个人实验推向团队生产的关键转折点。

CHAPTER THREE

第三章 模型生态完全指南

3.1 模型库与命名规则

3.2 量化级别完全手册

3.3 2026年5月推荐模型清单

3.4 模型选择决策树

3.1 模型库与命名规则

模型是 Ollama 的核心。截至 2026年5月，Ollama 官方库已收录超过 200 个优化模型，涵盖从 0.6B 到 400B+ 参数规模的全面选择。本章提供量化技术的深入解析、按场景分类的推荐模型清单，以及面向不同硬件配置的决策树，帮助您在质量、速度和资源消耗之间找到最佳平衡点。

Ollama 官方模型库地址：ollama.com/library，截至 2026年5月已收录 200+ 优化模型，涵盖文本、代码、视觉、嵌入、推理等多个类别。

命名规则

Ollama 模型标识符遵循以下格式：

```
[host/][namespace/]model[:tag]
```

组成部分	说明	示例
`host`	注册中心地址，省略时默认为 `registry.ollama.ai`	`hf.co`
`namespace`	组织或用户名称，官方模型使用 `library`（可省略）	`library/`
`model`	模型名称	`llama3.1`、`qwen3.5`
`tag`	版本标签，格式通常为 `参数量-量化级别` 或特定功能标识	`8b`、`14b-q5_K_M`、`latest`

完整示例： - `ollama run llama3.1:8b` — 官方 Llama 3.1 8B 参数模型，默认 Q4_K_M 量化 - `ollama run qwen3.5:14b-q5_K_M` — Qwen 3.5 14B，Q5_K_M 量化 - `ollama run hf.co/bartowski/Llama-3.2-3B-Instruct-GGUF:Q8_0` — 从 HuggingFace 直接拉取

3.2 量化级别完全手册

量化是本地运行大模型的核心技术。Ollama 使用 GGUF 格式，支持从 Q2_K 到 F16 的多种量化级别。

量化对比表

量化级别	精度	7B 模型显存	显存节省比	质量损失	推荐场景
F16	16-bit 浮点	~14 GB	0%（基准）	几乎无损	研究评测、追求极致质量
Q8_0	8-bit 整数	~7.4 GB	47%	几乎无损	高配设备，兼顾质量与性能
Q6_K	6-bit 混合	~6.1 GB	56%	极轻微	平衡选择，接近原始质量
Q5_K_M	5-bit 混合	~5.3 GB	62%	轻微	**日常主力，推荐首选**
Q5_K_S	5-bit 激进	~5.0 GB	64%	轻微+	显存略紧张时替代 Q5_K_M
Q4_K_M	4-bit 混合	~4.4 GB	69%	可接受	**通用默认，高性价比**
Q4_K_S	4-bit 激进	~4.1 GB	71%	可接受-	显存受限时的折中
Q3_K_M	3-bit 混合	~3.5 GB	75%	明显	低配设备，能跑优先
Q3_K_S	3-bit 激进	~3.2 GB	77%	明显	不推荐，除非显存极度不足
Q2_K	2-bit 混合	~2.7 GB	81%	严重	极限资源场景，质量损失显著

按场景的量化选择建议

使用场景	最低推荐量化	最佳量化	原因
代码生成 / 编程助手	Q5_K_M	Q8_0	代码对精度最敏感，低量化易产生语法错误
数学推理 / 逻辑分析	Q5_K_M	Q6_K / Q8_0	逻辑链条易被量化误差打断
日常对话 / 通用问答	Q4_K_M	Q5_K_M	对话场景对精度容忍度最高
创意写作 / 文案生成	Q4_K_M	Q5_K_M	创造性任务允许一定“模糊”
长文本摘要 / RAG	Q4_K_M	Q4_K_M	理解型任务对精度要求适中
嵌入式 / 边缘设备	Q3_K_M	Q4_K_M	受限于 4-8GB 显存

> **核心原则**：优先保证量化级别，再追求参数规模。7B Q5_K_M 的实际表现通常优于 13B Q2_K，因为过度量化会严重损害模型的基础能力。

3.3 2026年5月推荐模型清单

以下为截至 2026年5月，Ollama 官方库中可用且经过社区验证的最新模型。所有命令均可直接复制执行。

通用对话模型

模型名	参数量	推荐量化	显存需求	中文能力	ollama run 命令
Llama 4 Scout	17B (Active)	Q4_K_M	~10 GB	良好	`ollama run llama4:scout`
Llama 4 Maverick	17B (Active)	Q4_K_M	~10 GB	良好	`ollama run llama4:maverick`
Llama 3.3	70B	Q4_K_M	~40 GB	良好	`ollama run llama3.3:70b`
Qwen 3.5	0.6B-122B	Q4_K_M / Q5_K_M	按尺寸	**极强**	`ollama run qwen3.5`
Qwen 3	0.6B-30B	Q4_K_M	按尺寸	**极强**	`ollama run qwen3`
DeepSeek V3.2	16B-236B	Q4_K_M	~12-130 GB	强	`ollama run deepseek-v3.2`
Gemma 4	2B-31B	Q4_K_M	按尺寸	中等	`ollama run gemma4`
Gemma 3	1B-27B	Q4_K_M	按尺寸	中等	`ollama run gemma3`
Phi 4	14B	Q5_K_M	~10 GB	良好	`ollama run phi4`
Phi 4 Mini	3.8B	Q4_K_M / Q8_0	~3 GB	良好	`ollama run phi4-mini`
GLM-4.7-Flash	30B (MoE)	Q4_K_M	~16 GB	**极强**	`ollama run glm-4.7-flash`
GLM-5.1	9B-744B (MoE)	Q4_K_M	按尺寸	**极强**	`ollama run glm-5.1`
MiniMax-M2.1	32B (MoE)	Q4_K_M	~18 GB	强	`ollama run minimax-m2.1`
Mistral Large 3	123B	Q4_K_M	~75 GB	良好	`ollama run mistral-large:3`
Kimi K2.6	1T (MoE)	Q4_K_M	按激活参数	强	`ollama run kimi-k2.6`

> **说明**：Qwen 3.5 和 GLM-5.1 是当前中文能力最强的开源模型；Llama 4 系列在英文场景下表现卓越；Phi 4 Mini 以小博大，是低显存设备的首选。

代码模型

模型名	参数量	推荐量化	显存需求	中文能力	ollama run 命令
Qwen 3.5 Coder	7B-32B	Q5_K_M	按尺寸	强	`ollama run qwen3.5-coder`
Qwen 3 Coder	7B-30B	Q5_K_M	按尺寸	强	`ollama run qwen3-coder`
Qwen3-Coder-Next	专用版	Q5_K_M	~12 GB	强	`ollama run qwen3-coder-next`
DeepSeek Coder V2	16B-236B	Q5_K_M	按尺寸	强	`ollama run deepseek-coder-v2`
CodeLlama	7B-70B	Q4_K_M	按尺寸	一般	`ollama run codellama`
GPT-OSS	20B	Q4_K_M	~12 GB	良好	`ollama run gpt-oss`
Codestral	22B	Q5_K_M	~14 GB	一般	`ollama run codestral`

多模态视觉模型

模型名	参数量	推荐量化	显存需求	中文能力	ollama run 命令
Llama 4 Vision	17B (Active)	Q4_K_M	~12 GB	良好	`ollama run llama4:vision`
Llama 3.2 Vision	11B	Q4_K_M	~8 GB	良好	`ollama run llama3.2-vision`
Qwen 3 VL	3B-72B	Q4_K_M	按尺寸	**极强**	`ollama run qwen3-vl`
Qwen 2.5 VL	3B-72B	Q4_K_M	按尺寸	**极强**	`ollama run qwen2.5-vl`
Gemma 3 Vision	4B-27B	Q4_K_M	按尺寸	中等	`ollama run gemma3:vision`
Mistral Small 3	24B	Q4_K_M	~14 GB	一般	`ollama run mistral-small:3`

Embedding 模型

模型名	维度	推荐量化	显存需求	特点	ollama pull 命令
nomic-embed-text	768	Q4_K_M	~0.5 GB	轻量高效，RAG 首选	`ollama pull nomic-embed-text`
nomic-embed-text-v1.5	768	Q4_K_M	~0.5 GB	性能增强版	`ollama pull nomic-embed-text:v1.5`
bge-m3	1024	Q4_K_M	~2 GB	多语言支持优秀	`ollama pull bge-m3`
mxbai-embed-large	1024	Q4_K_M	~0.7 GB	高质量句子嵌入	`ollama pull mxbai-embed-large`
snowflake-arctic-embed	1024	Q4_K_M	~1 GB	长文本检索优化	`ollama pull snowflake-arctic-embed`
qwen3-embedding	1024	Q4_K_M	~1 GB	中文语义理解强	`ollama pull qwen3-embedding`
all-minilm	384	Q4_K_M	~0.1 GB	极致轻量，边缘设备	`ollama pull all-minilm`

推理思考模型

模型名	参数量	推荐量化	显存需求	特点	ollama run 命令
DeepSeek-R1	1.5B-70B	Q4_K_M / Q5_K_M	按尺寸	**最强开源推理**，思维链可见	`ollama run deepseek-r1`
Qwen 3 Thinking	0.6B-30B	Q4_K_M	按尺寸	内置思考模式，中文推理强	`ollama run qwen3`
QwQ	32B	Q5_K_M	~20 GB	阿里推理专优模型	`ollama run qwq`
SmallThinker	3B	Q4_K_M	~2 GB	轻量级推理，移动端可用	`ollama run smallthinker`

3.4 模型选择决策树

根据 **用途 × 硬件配置** 快速定位最优模型：

```
开始
│
├─ 用途：通用对话 / 问答
│   ├─ 显存 ≥ 40 GB → Llama 3.3 70B 或 Qwen 3.5 122B
│   ├─ 显存 16-24 GB → Qwen 3.5 32B / Gemma 4 27B / DeepSeek V3.2 16B
│   ├─ 显存 8-16 GB → Llama 3.1 8B / Qwen 3.5 14B / Gemma 4 4b-9b
│   └─ 显存 < 8 GB → Phi 4 Mini / Qwen 3 1.7B / Llama 3.2 3B
│
├─ 用途：代码生成 / 编程助手
│   ├─ 显存 ≥ 24 GB → Qwen 3.5 Coder 32B / DeepSeek Coder V2 236B
│   ├─ 显存 12-24 GB → Qwen 3 Coder 14B / Qwen3-Coder-Next
│   └─ 显存 < 12 GB → Qwen 2.5 Coder 7B / CodeLlama 7B / GPT-OSS 20B
│
├─ 用途：多模态（图文理解）
│   ├─ 显存 ≥ 16 GB → Qwen 3 VL 72B / Llama 4 Vision
│   ├─ 显存 8-16 GB → Llama 3.2 Vision 11B / Qwen 2.5 VL 7B
│   └─ 显存 < 8 GB → Gemma 3 Vision 4B / Qwen 3 VL 3B
│
├─ 用途：推理 / 数学 / 逻辑分析
│   ├─ 显存 ≥ 24 GB → DeepSeek-R1 70B
│   ├─ 显存 12-24 GB → DeepSeek-R1 32B / QwQ 32B
│   └─ 显存 < 12 GB → DeepSeek-R1 14B / Qwen 3 Thinking / SmallThinker
│
└─ 用途：文本嵌入 / RAG
    ├─ 中文为主 → qwen3-embedding / bge-m3
    └─ 英文为主 → nomic-embed-text / mxbai-embed-large
```

CHAPTER FOUR

第四章 Modelfile 与模型定制

4.1 Modelfile 语法完整参考

4.2 参数调优指南

4.3 导入外部模型

4.4 Modelfile 模板合集

4.1 Modelfile 语法完整参考

Modelfile 是 Ollama 的模型定义文件，语法受 Dockerfile 启发。通过 Modelfile，可以基于现有模型创建带有自定义系统提示、参数和上下文的衍生模型。

所有指令说明

指令	必填	说明	示例
`FROM`	是	基础模型源，可以是现有模型或 GGUF 文件路径	`FROM llama3.1:8b`
`PARAMETER`	否	设置推理参数	`PARAMETER temperature 0.7`
`TEMPLATE`	否	覆盖模型的对话模板（高级）	见 4.4 节
`SYSTEM`	否	系统提示词，定义模型角色与行为边界	`SYSTEM "你是一个翻译专家..."`
`ADAPTER`	否	加载 LoRA 适配器（.bin 格式）	`ADAPTER ./my-lora.bin`
`LICENSE`	否	指定模型许可证文本	`LICENSE "MIT"`
`MESSAGE`	否	预设多轮对话示例	`MESSAGE user "你好"`

完整示例

```
# 基于 Llama 3.1 8B 创建企业客服专用模型
FROM llama3.1:8b

SYSTEM """你是 TechCorp 智能客服助手。遵循以下原则：
1. 仅回答与 TechCorp 产品相关的问题
2. 遇到投诉先表达歉意再提供解决方案
3. 不知道的问题引导用户转人工服务（按 0）
4. 回复简洁，不超过 100 字"""

PARAMETER temperature 0.3
PARAMETER top_p 0.85
PARAMETER num_ctx 4096
PARAMETER num_predict 256
PARAMETER repeat_penalty 1.2

# 提供示例对话以引导行为
MESSAGE user "我的路由器连不上网了"
MESSAGE assistant "非常抱歉给您带来不便。请尝试以下步骤：1) 重启路由器电源 2) 检查网线连接 3) 确认宽带账户未欠费。如仍无法解决，请按 0 转人工。"

MESSAGE user "你们公司上市了吗"
MESSAGE assistant "这个问题不在我的服务范围内，请按 0 转人工咨询。"
```

创建并运行：

```
ollama create techcorp-custome -f Modelfile
ollama run techcorp-custome
```

4.2 参数调优指南

参数	作用	默认值	调优范围	建议
`temperature`	控制生成随机性，越低越保守	0.8	0.0 - 2.0	代码/事实问答用 0.1-0.3，创意写作用 0.8-1.2
`top_p`	核采样阈值，控制候选词范围	0.9	0.0 - 1.0	通常保持 0.85-0.95，与 temperature 配合调整
`top_k`	限制每步候选词数量	40	0 - 100	一般保持默认值，降低可使输出更保守
`num_ctx`	上下文窗口长度（Token 数）	2048	512 - 模型上限	长文档分析设为 8192-32768，注意显存占用随之增长
`num_predict`	最大生成 Token 数	-1（无限制）	-1 / 1 - 8192	限制输出长度用 128-512，防止过度生成
`repeat_penalty`	重复惩罚系数	1.1	1.0 - 2.0	遇到重复内容时提升至 1.15-1.3
`seed`	随机种子，固定值可复现输出	0（随机）	任意整数	调试或测试时固定 seed
`stop`	自定义停止序列	无	字符串列表	如 `["<|endoftext|>", "用户:"]`

4.3 导入外部模型

从 GGUF 文件导入

```
# 1. 准备 Modelfile
cat > Modelfile.gguf <<'EOF'
FROM ./my-model.Q4_K_M.gguf

SYSTEM "这是一个自定义导入模型。"
PARAMETER temperature 0.7
EOF

# 2. 创建模型
ollama create my-imported-model -f Modelfile.gguf

# 3. 验证
ollama list | grep my-imported-model
```

从 HuggingFace 导入（含国内镜像）

Ollama 支持直接从 HuggingFace 拉取 GGUF 模型：

```
# 官方 HuggingFace
ollama pull hf.co/bartowski/Llama-3.2-3B-Instruct-GGUF:Q8_0

# 国内镜像 hf-mirror.com（2026年5月已验证可用）
export HF_ENDPOINT=https://hf-mirror.com
ollama pull hf.co/bartowski/Llama-3.2-3B-Instruct-GGUF:Q8_0
```

huggingface-cli 配合国内镜像下载完整用法：

```
# 安装工具
pip install -U huggingface_hub

# 设置环境变量
export HF_ENDPOINT=https://hf-mirror.com

# 下载模型（支持断点续传）
huggingface-cli download bartowski/Llama-3.2-3B-Instruct-GGUF \
  --local-dir ./models/llama-3.2-3b \
  --resume-download

# 配合 hf_transfer 加速（10 倍下载速度）
```

```
pip install hf_transfer
export HF_HUB_ENABLE_HF_TRANSFER=1

huggingface-cli download bartowski/Llama-3.2-3B-Instruct-GGUF \
  --local-dir ./models/llama-3.2-3b
```

4.4 Modelfile 模板合集

编程助手模板

```
FROM qwen3.5-coder:14b

SYSTEM """你是一位全栈开发专家。遵循以下规范：
- 代码符合 PEP 8 / Google Style Guide
- 优先使用现代语言特性（Python 3.10+）
- 每个函数附带类型注解和 docstring
- 复杂逻辑添加中文注释
- 提供可运行的单元测试"""

PARAMETER temperature 0.2
PARAMETER top_p 0.9
PARAMETER num_ctx 16384
```

翻译专家模板

```
FROM qwen3.5:14b

SYSTEM """你是一位专业中英双语翻译。规则：
- 翻译准确传达原文含义，不遗漏信息
- 保持原文语体风格（正式/口语/文学）
- 技术术语保留英文并在首次出现时括号加注
- 长句适当拆分，确保中文流畅自然
- 仅输出翻译结果，不添加解释"""

PARAMETER temperature 0.3
PARAMETER num_predict 2048
```

创意写作模板

```
FROM llama4:scout

SYSTEM """你是一位创意写作助手。风格要求：
- 文字富有画面感和节奏感
- 善用隐喻和通感修辞
- 情节转折合乎逻辑又出人意料
- 对话自然，符合人物身份"""

PARAMETER temperature 1.0
```

```
PARAMETER top_p 0.95
PARAMETER repeat_penalty 1.15
```

客服机器人模板

```
FROM qwen3.5:7b

SYSTEM """你是企业客服代表。约束条件：
1. 语气礼貌、耐心、专业
2. 首次回复先确认客户问题
3. 提供步骤时编号列出，最多 3 步
4. 无法解决时提供人工通道
5. 绝不承诺超出权限的补偿方案"""

PARAMETER temperature 0.25
PARAMETER num_predict 150
```

第四章总结：Modelfile 是 Ollama 从 "模型运行工具" 升级为 "模型定制平台" 的核心。通过合理组合 SYSTEM 指令、PARAMETER 调优和示例 MESSAGE，您可以为特定业务场景打造出表现远超通用模型的专用 AI。记住：好的模型定制 = 清晰的角色定义 + 恰当的参数约束 + 充分的示例引导。

CHAPTER FIVE

第五章 API 完全手册

5.1 原生 API

5.2 OpenAI 兼容 API（重点）

5.3 结构化输出

5.4 工具调用（Function Calling）

5.5 多模态输入与流式输出

5.1 原生 API

Ollama 的 REST API 是其与外部世界交互的主要接口。原生 API 提供了对模型推理的细粒度控制，而 OpenAI 兼容 API 则确保了与现有工具链的零摩擦集成。本章通过完整的请求响应示例，覆盖从基础文本生成到高级工具调用的全部能力。

Ollama 默认监听 `http://localhost:11434`，所有 API 端点均返回 JSON。

`/api/generate` — 文本生成

完整请求响应示例：

```
curl http://localhost:11434/api/generate -d '{
  "model": "llama3.1:8b",
  "prompt": "为什么天空是蓝色的？",
  "system": "用简洁的语言回答。",
  "stream": false,
  "options": {
    "temperature": 0.7,
    "num_predict": 256
  }
}'
```

响应：

```
{
  "model": "llama3.1:8b",
  "created_at": "2026-05-01T08:30:00Z",
  "response": "天空呈现蓝色是因为瑞利散射—大气分子对短波长蓝光散射强于长波长红光。",
  "done": true,
  "context": [128006, 128009, ...],
  "total_duration": 1450000000,
  "load_duration": 300000000,
  "prompt_eval_duration": 50000000,
  "eval_duration": 1100000000,
  "prompt_eval_count": 12,
  "eval_count": 45
}
```

`/api/chat` — 多轮对话

```
curl http://localhost:11434/api/chat -d '{
  "model": "qwen3.5:14b",
  "messages": [
    {"role": "system", "content": "你是 helpful assistant。"},
    {"role": "user", "content": "解释 TCP 三次握手"},
    {"role": "assistant", "content": "TCP 三次握手是建立连接的过程..."},
    {"role": "user", "content": "那四次挥手呢？"}
  ],
  "stream": false,
  "options": {
    "temperature": 0.5
  }
}'
```

`/api/embed` — 向量生成

```
curl http://localhost:11434/api/embed -d '{
  "model": "nomic-embed-text",
  "input": "大语言模型本地部署的优势"
}'
```

响应包含 `embeddings` 数组，每个元素为浮点向量。

其他管理端点

端点	方法	说明	示例请求体
`/api/tags`	GET	列出本地模型	无
`/api/show`	POST	查看模型详情	`{"model": "llama3.1:8b"}`
`/api/pull`	POST	拉取模型	`{"model": "qwen3.5"}`
`/api/delete`	DELETE	删除模型	`{"model": "old-model"}`
`/api/copy`	POST	复制模型	`{"source": "a", "destination": "b"}`
`/api/ps`	GET	查看运行中的模型	无
`/api/version`	GET	查看 Ollama 版本	无

`/api/tags` 等管理端点使用示例

```
# 列出所有本地模型
curl http://localhost:11434/api/tags | jq '.models[] | .name'

# 查看特定模型详情
curl http://localhost:11434/api/show -d '{"model": "qwen3.5:14b"}' | jq '.details'

# 删除模型
curl -X DELETE http://localhost:11434/api/delete -d '{"model": "old-model:7b"}'

# 复制模型（创建别名）
curl http://localhost:11434/api/copy -d '{
  "source": "llama3.1:8b",
  "destination": "production-chat:8b"
}'
```

这些管理端点对于自动化运维和 CI/CD 集成非常重要。例如，可以在部署流水线中通过 `/api/pull` 确保生产环境拥有最新模型版本，通过 `/api/ps` 监控服务负载，通过 `/api/delete` 清理过期模型以节省磁盘空间。

5.2 OpenAI 兼容 API（重点）

Ollama 提供与 OpenAI REST API 格式兼容的端点，路径为 `/v1/chat/completions`、`/v1/completions`、`/v1/embeddings` 和 `/v1/models`。

兼容端点表

OpenAI 端点	Ollama 兼容端点	状态
`POST /v1/chat/completions`	`POST /v1/chat/completions`	完整支持，含流式输出、工具调用
`POST /v1/completions`	`POST /v1/completions`	支持（Legacy）
`POST /v1/embeddings`	`POST /v1/embeddings`	完整支持
`GET /v1/models`	`GET /v1/models`	完整支持

Python SDK 调用完整示例

```
from openai import OpenAI

client = OpenAI(
    base_url="http://localhost:11434/v1",
    api_key="ollama"  # Ollama 不校验 key，但 SDK 要求必填
)

# 对话请求
response = client.chat.completions.create(
    model="qwen3.5:14b",
    messages=[
        {"role": "system", "content": "你是专业程序员。"},
        {"role": "user", "content": "用 Python 写快速排序"}
    ],
    temperature=0.2,
    stream=False
)
print(response.choices[0].message.content)

# 流式请求
stream = client.chat.completions.create(
    model="llama3.1:8b",
    messages=[{"role": "user", "content": "讲个笑话"}],
    stream=True
)
```

```
for chunk in stream:
    if chunk.choices[0].delta.content:
        print(chunk.choices[0].delta.content, end="", flush=True)

# 嵌入请求
embeddings = client.embeddings.create(
    model="nomic-embed-text",
    input=["文本一", "文本二"]
)
print(embeddings.data[0].embedding)
```

Node.js SDK 调用完整示例

```
import OpenAI from 'openai';

const client = new OpenAI({
  baseURL: 'http://localhost:11434/v1',
  apiKey: 'ollama'
});

async function chat() {
  const response = await client.chat.completions.create({
    model: 'gemma4:e4b',
    messages: [
      { role: 'system', content: '你是数据分析助手。' },
      { role: 'user', content: '解释 Pandas groupby 用法' }
    ],
    temperature: 0.3
  });
  console.log(response.choices[0].message.content);
}

async function streamChat() {
  const stream = await client.chat.completions.create({
    model: 'qwen3.5:14b',
    messages: [{ role: 'user', content: '写一首关于 AI 的诗' }],
    stream: true
  });
  for await (const chunk of stream) {
    process.stdout.write(chunk.choices[0]?.delta?.content || '');
  }
}

chat();
```

5.3 结构化输出

JSON Mode 基础用法

```
curl http://localhost:11434/api/chat -d '{
  "model": "qwen3.5:14b",
  "messages": [{"role": "user", "content": "提取姓名、年龄、城市"}],
  "format": "json",
  "stream": false
}'
```

> **提示**：使用 JSON Mode 时，务必在 prompt 中明确要求模型输出 JSON 格式，并给出键名示例。

JSON Schema 约束（v0.18+ 新功能）

Ollama 支持通过 JSON Schema 严格约束输出结构：

```
curl http://localhost:11434/api/chat -d '{
  "model": "llama3.1:8b",
  "messages": [{"role": "user", "content": "生成一个用户信息"}],
  "format": {
    "type": "object",
    "properties": {
      "name": {"type": "string"},
      "age": {"type": "integer"},
      "email": {"type": "string"},
      "tags": {"type": "array", "items": {"type": "string"}}
    },
    "required": ["name", "age", "email"]
  }
}'
```

通过 Schema 约束，可在 Agent 工作流中实现可靠的输出解析，消除 retry 循环。

5.4 工具调用（Function Calling）

工具调用是现代 AI Agent 的核心能力，它允许模型在需要时调用外部函数获取数据或执行操作，再将结果整合到回复中。Ollama 从 v0.20 开始完整支持 OpenAI 格式的工具调用协议。

支持模型列表

模型	工具调用支持
Llama 3.1+	原生支持
Llama 4	原生支持
Qwen 3 / 3.5	原生支持
Mistral Large	原生支持
DeepSeek V2.5+	原生支持
Gemma 4	原生支持

完整请求响应示例

```
curl http://localhost:11434/v1/chat/completions -d '{
  "model": "qwen3.5:14b",
  "messages": [{"role": "user", "content": "北京今天天气怎么样？"}],
  "tools": [
    {
      "type": "function",
      "function": {
        "name": "get_weather",
        "description": "获取指定城市的当前天气",
        "parameters": {
          "type": "object",
          "properties": {
            "city": {"type": "string", "description": "城市名称"},
            "unit": {"type": "string", "enum": ["celsius", "fahrenheit"]}
          },
          "required": ["city"]
        }
      }
    }
  ],
```

```
  "tool_choice": "auto"
}'
```

响应（当模型决定调用工具时）：

```
{
  "choices": [{
    "message": {
      "role": "assistant",
      "tool_calls": [{
        "id": "call_abc123",
        "type": "function",
        "function": {
          "name": "get_weather",
          "arguments": "{\"city\": \"北京\", \"unit\": \"celsius\"}"
        }
      }]
    }
  }]
}
```

工具调用完整工作流

工具调用（Function Calling）的典型应用流程包括以下步骤：

1. **定义工具 Schema**：在 API 请求中通过 `tools` 参数描述可用工具的名称、功能和参数格式
2. **模型决策**：模型根据用户输入判断是否需要调用工具，若需要则返回 `tool_calls` 而非直接文本
3. **执行工具**：客户端解析 `tool_calls`，执行对应的真实工具函数，获取结果
4. **反馈结果**：将工具执行结果以 `tool` 角色的消息追加到对话历史，再次请求模型
5. **模型综合**：模型基于工具返回数据生成最终回复

以下是一个完整的 Python 示例，展示了天气查询 agent 的实现：

```
from openai import OpenAI
import json

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

def get_weather(city, unit="celsius"):
    return {"city": city, "temperature": 22, "condition": "晴", "unit": unit}

messages = [{"role": "user", "content": "北京今天天气怎么样？"}]
tools = [{
    "type": "function",
    "function": {
        "name": "get_weather",
        "description": "获取城市天气",
```

```
        "parameters": {
            "type": "object",
            "properties": {
                "city": {"type": "string"},
                "unit": {"type": "string", "enum": ["celsius", "fahrenheit"]}
            },
            "required": ["city"]
        }
    }
}]

# 第一次请求：模型决定是否调用工具
resp = client.chat.completions.create(
    model="qwen3.5:14b", messages=messages, tools=tools, tool_choice="auto"
)
msg = resp.choices[0].message

# 若需调用工具
if msg.tool_calls:
    messages.append({"role": "assistant", "tool_calls": [tc.model_dump() for tc in msg.tool_calls]})
    for tc in msg.tool_calls:
        fn_name = tc.function.name
        args = json.loads(tc.function.arguments)
        result = get_weather(**args)
        messages.append({"role": "tool", "tool_call_id": tc.id, "content": json.dumps(result)})

    # 第二次请求：模型综合工具结果生成回复
    resp2 = client.chat.completions.create(model="qwen3.5:14b", messages=messages)
    print(resp2.choices[0].message.content)
```

5.5 多模态输入与流式输出

多模态输入

```
# 对图片进行描述
curl http://localhost:11434/api/chat -d '{
  "model": "qwen3-vl:7b",
  "messages": [{
    "role": "user",
    "content": "描述这张图片",
    "images": ["/9j/4AAQSkZJRgABAQAAAQ..."]
  }]
}'
```

图片需以 Base64 编码传入 `images` 数组。支持 JPG、PNG 格式。

流式输出

所有聊天和生成 API 均支持 SSE（Server-Sent Events）流式输出。设置 `"stream": true` 即可逐 Token 接收响应：

```
curl http://localhost:11434/api/chat -d '{
  "model": "llama3.1:8b",
  "messages": [{"role": "user", "content": "讲一个科幻故事开头"}],
  "stream": true
}'
```

每行返回一个 JSON 对象，包含 `message.content` 片段。最后一行的 `done` 字段为 `true`。

流式输出对于交互式应用至关重要。当模型生成较长的回复时，流式传输可以让用户立即看到前几个字的输出，而不是等待数十秒后的完整响应。在 Web 应用中，通常配合 Server-Sent Events (SSE) 或 WebSocket 实现打字机效果。Python 的 `openai` SDK 通过 `stream=True` 参数自动处理流式响应解析，开发者只需迭代 `for chunk in stream` 即可逐 Token 消费输出，大幅提升了用户体验的流畅度。

CHAPTER SIX

第六章 性能优化完全指南

6.1 GPU 加速

6.2 推理速度优化

6.3 内存与存储优化

6.4 性能基准测试

6.1 GPU 加速

性能优化是将 Ollama 从 "能用" 推向 "好用" 的关键环节。本章覆盖 GPU 加速配置、推理速度优化、内存管理和基准测试方法论。合理应用这些技巧，可在相同硬件上获得 2-5 倍的吞吐量提升。

Flash Attention 启用方法及效果

Flash Attention 是现代 Transformer 模型的关键优化技术，可显著降低长上下文内存占用并提升速度。

```
# 启用 Flash Attention
export OLLAMA_FLASH_ATTENTION=1
ollama serve
```

效果：在 4096+ Token 的长上下文场景中，显存占用降低 30-50%，推理速度提升 10-20%。短上下文场景收益较小。

KV Cache 量化

KV Cache 是长上下文推理的主要显存消耗源。Ollama 支持对其进行量化：

```
export OLLAMA_KV_CACHE_TYPE=q8_0   # 推荐：显存减半，质量几乎无损
# 或
export OLLAMA_KV_CACHE_TYPE=q4_0   # 显存降至 1/4，长上下文时可能有轻微损失
```

KV Cache 量化	显存占用	质量影响	推荐场景
f16（默认）	100%	无	短上下文、显存充裕
q8_0	~50%	几乎无损	**长上下文首选**
q4_0	~25%	轻微	极限长上下文（32K+）

GPU 层数控制

`OLLAMA_NUM_GPU` 控制加载到 GPU 的模型层数。默认自动分配，手动设置可用于多卡或部分卸载：

```
# 全部加载到 GPU
export OLLAMA_NUM_GPU=99
```

```
# 仅加载 20 层到 GPU，其余在 CPU（用于显存不足时运行大模型）
export OLLAMA_NUM_GPU=20
```

NVIDIA / AMD / Apple Silicon 各自优化参数

平台	优化参数 / 说明
NVIDIA	确保驱动 ≥535，CUDA 12.x。使用 `OLLAMA_FLASH_ATTENTION=1` 收益最大
AMD	Linux + ROCm 6.0+。Windows 下 AMD GPU 当前仅支持 CPU 模式（2026年5月）
Apple Silicon	Ollama 通过 MLX 后端自动优化 Metal。M4 系列比 M3 快 15-25%。建议保持默认配置

各平台 GPU 加速详细配置

NVIDIA CUDA（Linux / Windows）：

确保 NVIDIA 驱动版本 ≥535，CUDA Toolkit ≥12.0。可通过以下命令验证：

```
nvidia-smi  # 查看驱动版本和 GPU 状态
nvcc --version  # 查看 CUDA 版本
```

对于多 GPU 工作站，Ollama 默认会将模型完整地加载到单张 GPU 上以最大化性能。若模型体积超过单卡显存，则会自动将模型层分散到多张 GPU。可通过 `CUDA_VISIBLE_DEVICES` 环境变量控制可见 GPU。

AMD ROCm（仅 Linux）：

Ollama 通过 ROCm 后端支持 AMD GPU。截至 2026年5月，ROCm 7.2.1 已集成到 Ollama 官方发行版中。支持的 GPU 包括 MI 系列数据中心卡和 RX 7000 系列消费级显卡。Windows 平台目前尚无原生 ROCm 支持，AMD GPU Windows 用户建议直接使用 CPU 模式或等待 Vulkan 后端正式推出。

Apple Silicon Metal：

Ollama 在 M 系列芯片上自动使用 Apple 的 Metal Performance Shaders 进行加速。v0.21.x 引入了多项 MLX 优化，包括 fused top-P/top-K sampling 和 faster tokenization。Apple Silicon 用户通常不需要手动调整任何 GPU 参数，保持默认配置即可获得最优性能。

6.2 推理速度优化

优化手段表

优化项	配置方法	预期提升	适用条件
Flash Attention	`OLLAMA_FLASH_ATTENTION=1`	10-50%（长上下文更明显）	所有支持平台
KV Cache 量化	`OLLAMA_KV_CACHE_TYPE=q8_0`	释放显存用于更大批处理	长上下文场景
量化模型	使用 Q4_K_M 替代 F16	模型体积减半，加载更快	显存受限时
并行请求	`OLLAMA_NUM_PARALLEL=4`	吞吐量提升 3-4 倍	多用户并发
模型预热	启动时预先加载模型	消除首次请求加载延迟	生产环境

生产环境推荐配置

```
# 服务器环境变量配置示例
export OLLAMA_HOST=0.0.0.0:11434
export OLLAMA_NUM_PARALLEL=4
export OLLAMA_MAX_LOADED_MODELS=2
export OLLAMA_KEEP_ALIVE=-1          # 永久驻留，避免重复加载
export OLLAMA_FLASH_ATTENTION=1
export OLLAMA_KV_CACHE_TYPE=q8_0
export OLLAMA_NUM_GPU=99
```

6.3 内存与存储优化

模型存储路径自定义

默认模型存储位置： - macOS: `~/.ollama/models` - Linux: `/usr/share/ollama/.ollama/models` - Windows: `C:\Users\<用户名>\.ollama\models`

修改存储路径：

```
# Linux / macOS
export OLLAMA_MODELS=/mnt/nvme/ollama-models

# Windows（系统环境变量）
OLLAMA_MODELS=D:\AI_Models
```

> **注意**：修改后需重启 Ollama 服务。已有模型不会自动迁移，需手动移动或重新拉取。

ollama prune 清理

```
# 清理所有未使用的模型层（释放磁盘空间）
ollama prune
```

KEEP_ALIVE 策略

`OLLAMA_KEEP_ALIVE` 控制模型在内存中的驻留时间：

设置值	行为	适用场景
`0`	请求结束后立即卸载	显存极度紧张，单次调用
`5m`	空闲 5 分钟后卸载	开发测试，平衡响应与资源
`-1`	永久驻留直到手动停止	生产服务，追求最低延迟

6.4 性能基准测试

测试生成速度的简易方法：

```
# 使用 time 命令测量端到端时间
time ollama run llama3.1:8b "写一段 500 字的科幻故事" --keepalive 0

# 使用 API 获取详细的 Token 级指标
curl http://localhost:11434/api/generate -d '{
  "model": "llama3.1:8b",
  "prompt": "解释机器学习",
  "stream": false
}' | jq '{prompt_eval_count, eval_count, eval_duration, tokens_per_sec: (.eval_count / (.eval_duration / 1e9))}'
```

第六章总结：性能优化的核心原则是"先测量，后优化"。`OLLAMA_FLASH_ATTENTION=1` 和 `OLLAMA_KV_CACHE_TYPE=q8_0` 是最具性价比的两项配置，几乎适用于所有场景。对于生产环境，`OLLAMA_KEEP_ALIVE=-1` 配合 `OLLAMA_MAX_LOADED_MODELS` 的合理设置能够显著改善首次响应延迟。

CHAPTER SEVEN

第七章 生态集成

7.1 桌面客户端

7.2 开发工具集成

7.3 自动化平台

7.4 知识库与 RAG

7.5 MCP 生态集成

7.1 桌面客户端

Ollama 本身不提供图形界面，但其开放的 API 架构催生了繁荣的第三方客户端生态。这些客户端覆盖了从个人聊天到企业知识库、从代码编程到自动化工作流的全部场景。选择合适的客户端可以大幅提升使用体验，降低非技术用户的上手门槛。

Ollama 的强大不仅在于其推理能力，更在于其与庞大工具生态的无缝集成。从精美的桌面聊天客户端到 IDE 编程助手，从可视化工作流自动化到企业级 RAG 知识库，Ollama 作为标准化的本地推理后端，正在连接起一个完整的私有化 AI 应用矩阵。

客户端	平台	特点	Ollama 配置方式
Cherry Studio	Win/Mac/Linux	国产精品，多服务商聚合，UI 精美	设置 → 模型 → 添加 Ollama 服务，输入 `http://localhost:11434`
Open WebUI	Web / Docker	功能最全的 Web 界面，支持 RAG、多用户	Docker Compose 一键部署，`OLLAMA_BASE_URL` 指向 Ollama
Jan	Win/Mac/Linux	100% 离线运行，隐私优先，界面现代	设置 → 扩展 → 连接本地 Ollama
Chatbox	Win/Mac/Linux	轻量简洁，跨平台	设置 → API → 选择 Ollama，输入本地地址
Enchanted	macOS / iOS	Swift 原生，Apple 生态体验极佳	自动发现本地 Ollama 实例

Cherry Studio 配置详细说明

1. 下载并安装 Cherry Studio（cherry-ai.com）
2. 打开应用，点击左下角 **设置**（齿轮图标）
3. 选择 **模型服务** → 点击 **Ollama**
4. 开启开关，**API 地址**填入 `http://localhost:11434`
5. 点击 **获取模型列表**，勾选需要显示的模型
6. 返回对话界面，在顶部模型下拉框选择 Ollama 模型即可开始对话

> 若 Cherry Studio 无法连接，检查 Ollama 是否设置了 `OLLAMA_HOST=0.0.0.0` 并允许跨域：`OLLAMA_ORIGINS=*`

7.2 开发工具集成

Continue（VS Code / JetBrains）

Continue 是开源的 AI 编程助手，完美支持 Ollama 后端。

`config.json` 示例：

```
{
  "models": [
    {
      "title": "Ollama Qwen Coder",
      "provider": "ollama",
      "model": "qwen3.5-coder:14b",
      "apiBase": "http://localhost:11434"
    }
  ],
  "tabAutocompleteModel": {
    "title": "Ollama Code Complete",
    "provider": "ollama",
    "model": "qwen3.5-coder:7b",
    "apiBase": "http://localhost:11434"
  }
}
```

Cursor

Cursor 内置支持 OpenAI 兼容 API。设置 → Models → OpenAI API Key → Base URL 填入 `http://localhost:11434/v1`，API Key 填任意值（如 `ollama`）。

Cody（Sourcegraph）

Cody 可配置自定义模型端点。在设置中找到 **Experimental** → **Custom Model**，输入 Ollama 的 OpenAI 兼容端点。

Aider

Aider 是终端里的 AI 结对编程工具：

```
export OLLAMA_API_BASE=http://localhost:11434/v1
aider --model ollama/qwen3.5-coder:14b
```

7.3 自动化平台

n8n（Ollama Chat Model 节点）

在 n8n 工作流中，添加 **Ollama Chat Model** 节点： - **Base URL**: `http://localhost:11434` - **Model**: `qwen3.5:14b` - 直接接入 AI Agent、If 节点、HTTP 请求等构建自动化流程

Dify（模型提供者配置）

Dify 是开源 LLM 应用开发平台： 1. 进入 **设置** → **模型提供商** 2. 点击 **Ollama** 卡片 3. **模型名称**填入 `qwen3.5:14b`，**Base URL**填入 `http://host.docker.internal:11434`（若 Dify 也在 Docker 中） 4. 保存后可在应用构建中直接选用

LangChain（Python）

```
from langchain_ollama import ChatOllama, OllamaEmbeddings

llm = ChatOllama(model="llama3.1:8b", base_url="http://localhost:11434")
embeddings = OllamaEmbeddings(model="nomic-embed-text", base_url="http://localhost:11434")

response = llm.invoke("什么是 RAG？")
print(response.content)
```

LlamaIndex

```
from llama_index.llms.ollama import Ollama
from llama_index.embeddings.ollama import OllamaEmbedding

llm = Ollama(model="qwen3.5:14b", base_url="http://localhost:11434")
embed_model = OllamaEmbedding(model_name="nomic-embed-text", base_url="http://localhost:11434")
```

7.4 知识库与 RAG

工具	特点	Ollama 集成方式
AnythingLLM	企业级，多工作区，权限管理	设置 → AI Provider → Ollama
PrivateGPT	100% 本地隐私，无需联网	原生支持 Ollama 作为后端
Quivr	开源 Second Brain，支持多格式	配置 `.env` 指向 Ollama 端点
LibreChat	类 ChatGPT 界面，多模型切换	设置 → 端点 → Ollama

7.5 MCP 生态集成

MCP（Model Context Protocol）是 2026 年迅速普及的开放协议，用于标准化模型与外部工具、数据源之间的通信。Ollama 作为本地推理后端，可通过 MCP Server 为支持 MCP 的客户端提供能力。

使用场景： - **Claude Desktop + Ollama MCP**：在 Claude Desktop 中调用本地 Ollama 模型处理敏感数据 - **OpenClaw + Ollama**：通过 `ollama launch openclaw` 一键部署本地 Agent 框架 - **自定义 MCP Server**：基于 `mcp` SDK 开发专用服务，将 Ollama 嵌入企业工作流

```
# OpenClaw 一键启动（需已安装 Node.js）
ollama launch openclaw --model qwen3.5:14b
```

第七章总结：生态集成是 Ollama 价值的放大器。个人用户可通过 Cherry Studio 或 Open WebUI 获得出色的交互体验，开发者通过 Continue.dev 和 LangChain 将 AI 嵌入工作流，企业团队通过 Dify 和 n8n 构建自动化业务系统。MCP 协议的兴起进一步打破了工具之间的壁垒，使 Ollama 成为 Agent 时代的基础设施核心。

CHAPTER EIGHT

第八章 企业级部署方案

8.1 单机多卡部署

8.2 负载均衡与高可用

8.3 监控与日志

8.4 安全加固

8.1 单机多卡部署

当 Ollama 从个人工作站走向团队共享或生产服务时，单机性能、高可用架构和安全合规成为必须面对的课题。本章提供经过验证的企业级部署方案，帮助技术团队构建稳定、安全、可扩展的私有 AI 基础设施。

Ollama 自动检测多 GPU 环境。加载模型时的策略如下：

1. **单卡能装下**：优先将整个模型加载到单张 GPU，避免跨卡通信开销
2. **单卡装不下**：自动将模型层分布到所有可用 GPU
3. **多卡装不下**：GPU 加载部分层，剩余层卸载至 CPU 内存

手动控制多卡分配（高级）：

```
# 指定仅使用第 0、1 号 GPU
export CUDA_VISIBLE_DEVICES=0,1
ollama serve
```

8.2 负载均衡与高可用

多实例 + Nginx 反向代理配置

```
# /etc/nginx/conf.d/ollama.conf
upstream ollama_backend {
    least_conn;                    # 最少连接负载均衡
    server 192.168.1.10:11434;     # Ollama 实例 1
    server 192.168.1.11:11434;     # Ollama 实例 2
    server 192.168.1.12:11434 backup;  # 备份实例
    keepalive 32;
}

server {
    listen 443 ssl;
    server_name ai.company.internal;

    ssl_certificate /etc/ssl/certs/company.crt;
    ssl_certificate_key /etc/ssl/private/company.key;

    # 基础认证
    auth_basic "AI Gateway";
    auth_basic_user_file /etc/nginx/.htpasswd;

    location / {
        proxy_pass http://ollama_backend;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_buffering off;          # 流式响应必需
        proxy_read_timeout 300s;      # 长生成超时
        proxy_http_version 1.1;
        proxy_set_header Connection "";
    }
}
```

多实例部署架构建议

在生产环境中，单实例 Ollama 往往无法满足高并发需求。推荐的横向扩展架构如下：

1. **模型分发层**：使用共享存储（NFS 或云盘）作为 `OLLAMA_MODELS` 目录，确保所有实例访问相同的模型文件
2. **负载均衡层**：Nginx 的 `least_conn` 策略将请求路由到当前连接数最少的实例，避免单点过载
3. **健康检查**：配置 Nginx 主动健康检查（需 `nginx-plus` 或第三方模块），自动剔除故障实例

4. **会话亲和性**：若需要保持对话上下文一致性，可通过 cookie 或 IP hash 将同一用户路由到同一实例

Docker Swarm 部署示例

```
version: '3.8'
services:
  ollama:
    image: ollama/ollama:latest
    deploy:
      replicas: 3
      placement:
        constraints: [node.labels.gpu == true]
    environment:
      - OLLAMA_NUM_PARALLEL=4
      - OLLAMA_FLASH_ATTENTION=1
    volumes:
      - ollama_shared:/root/.ollama
volumes:
  ollama_shared:
    driver: nfs
    driver_opts:
      share: "192.168.1.5:/exports/ollama"
```

8.3 监控与日志

内置监控端点

```
# 查看运行中的模型及资源占用
curl http://localhost:11434/api/ps
```

响应包含各模型的内存占用、GPU 使用率、处理器分配比例。

日志路径

平台	日志路径
macOS	`~/.ollama/logs/server.log`
Linux	`~/.ollama/logs/server.log` 或 journald
Windows	`%LOCALAPPDATA%\Ollama\logs\server.log`

实时查看：

```
tail -f ~/.ollama/logs/server.log
```

Prometheus + Grafana（可选）

通过轻量级 exporter 将 `/api/ps` 数据转为 Prometheus 指标，实现 GPU 利用率、请求延迟、并发量等维度的可视化监控。

8.4 安全加固

1. **网络隔离**：生产环境禁止 `OLLAMA_HOST=0.0.0.0` 无认证暴露。必须配合防火墙或反向代理的 Basic Auth
2. **TLS 加密**：通过 Nginx 或 Traefik 终止 TLS，禁止明文 HTTP 跨公网传输
3. **CORS 限制**：设置 `OLLAMA_ORIGINS=https://app.company.com`，而非 `*`
4. **模型拉取审计**：监控 `/api/pull` 调用，防止内部人员加载未授权模型
5. **定期更新**：关注 Ollama Security Advisories，及时升级修复 CVE

> **安全警告**：默认情况下 Ollama 仅监听 127.0.0.1。若修改为 `0.0.0.0`，任何能访问该端口的设备都可无认证调用模型 API 并加载任意模型，存在严重安全隐患。

第八章总结：企业级 Ollama 部署的核心在于"冗余、隔离、监控"三要素。Nginx 负载均衡确保服务不中断，Docker Swarm 提供弹性伸缩，环境变量与 CORS 配置构筑安全边界。配合 Prometheus 监控和结构化日志，技术团队可以获得与云 API 相媲美的服务稳定性。

CHAPTER NINE

第九章 常见问题排查

1. GPU 不可用 / Ollama 仅使用 CPU
2. 内存不足 / 模型加载失败
3. 模型下载慢 / 国内镜像方案
4. 端口冲突
5. 模型加载失败 / GGUF 不兼容
6. 推理速度慢
7. API 调用超时
8. 中文输出乱码
9. Windows WSL2 无法连接宿主机 Ollama
10. 模型回答质量差 / 不符合预期

1. GPU 不可用 / Ollama 仅使用 CPU

本章汇总了社区和团队在生产环境中最常遇到的 10 个问题。每个问题均包含现象描述、根因分析和经过验证的解决步骤。建议读者在遇到问题时先定位最接近的症状，再按步骤逐一排查。若以下方案均无法解决，可查阅 Ollama 官方 GitHub Issues（github.com/ollama/ollama/issues）获取社区支持。

即使是最成熟的系统也难免遇到问题。本章基于 2026年5月最新版本和大量社区反馈，整理了最具代表性的 10 个常见问题。每个问题的排查流程均按照"验证现象 → 定位根因 → 执行修复 → 确认恢复"的结构组织，力求让读者在最短时间内恢复正常使用。

现象：`ollama ps` 显示 `PROCESSOR` 为 `100% CPU`，或生成速度极慢（<5 tok/s）。

原因分析： - NVIDIA：驱动版本过低、CUDA 未正确安装、Docker 未配置 NVIDIA Container Toolkit - AMD：Windows 下 ROCm 不支持；Linux 下 ROCm 未安装 - Apple Silicon：正常应自动使用 Metal，若异常可能是 macOS 版本过低

解决步骤：

```
# 1. 验证 NVIDIA 驱动
nvidia-smi

# 2. 检查 CUDA 版本（Linux）
nvcc --version

# 3. Docker 环境需确认 toolkit
 docker run --rm --gpus all nvidia/cuda:12.0-base-ubuntu22.04 nvidia-smi

# 4. 若仍不行，手动指定 GPU 层数
export OLLAMA_NUM_GPU=99
ollama serve

# 5. 查看日志中的 GPU 检测信息
grep -i "gpu\|cuda\|metal" ~/.ollama/logs/server.log
```

2. 内存不足 / 模型加载失败

现象：拉取模型成功，但运行时报错 `out of memory` 或进程被 kill。

原因分析：模型体积超过可用 RAM/VRAM，或系统其他进程占用过多内存。

解决步骤：

```
# 1. 确认模型所需显存
ollama show llama3.3:70b --modelfile | grep -i size

# 2. 换用更小模型或更高量化级别
ollama run llama3.1:8b          # 替代 70B
ollama run qwen3.5:4b            # 极致轻量

# 3. 启用 KV Cache 量化节省显存
export OLLAMA_KV_CACHE_TYPE=q8_0

# 4. 减少上下文长度
# 在 Modelfile 或 API 请求中设置："options": {"num_ctx": 2048}

# 5. Linux 下关闭内存过度分配（可选）
sudo sysctl vm.overcommit_memory=2
```

3. 模型下载慢 / 国内镜像方案

现象：`ollama pull` 速度极慢或频繁中断。

原因分析：Ollama 官方库部分模型托管于海外 CDN，国内直连带宽受限。

解决步骤：

```
# 方案 1：使用代理（如已有 VPN）
export HTTPS_PROXY=http://127.0.0.1:7890
ollama pull qwen3.5

# 方案 2：通过 hf-mirror.com 拉取 HuggingFace 模型
export HF_ENDPOINT=https://hf-mirror.com
ollama pull hf.co/<namespace>/<model>-GGUF:Q4_K_M

# 方案 3：手动下载后导入
# 使用 hf-mirror.com 下载 GGUF，再通过 FROM 导入 Modelfile
```

4. 端口冲突

现象：启动 Ollama 时提示 `address already in use`。

解决步骤：

```
# 1. 查找占用 11434 的进程
lsof -i :11434
# 或
ss -tlnp | grep 11434

# 2. 终止冲突进程，或修改 Ollama 端口
export OLLAMA_HOST=127.0.0.1:11435
ollama serve
```

5. 模型加载失败 / GGUF 不兼容

现象：运行模型时崩溃或报错 `unknown model architecture`。

原因分析：GGUF 文件版本过旧，或模型架构尚未被当前 Ollama 版本支持。

解决步骤：

```
# 1. 升级 Ollama 至最新版
curl -fsSL https://ollama.com/install.sh | sh

# 2. 确认模型在官方支持列表中
# 访问 https://ollama.com/library 搜索模型名称

# 3. 若使用自定义 GGUF，确保转换为最新 GGUF 版本
# 使用 llama.cpp 的 convert.py 或 quantize 工具重新转换
```

6. 推理速度慢

现象：生成速度远低于预期（如 7B 模型 <10 tok/s）。

解决步骤：

```
# 1. 确认正在使用 GPU
ollama ps

# 2. 启用 Flash Attention
export OLLAMA_FLASH_ATTENTION=1

# 3. 使用更高量化级别（低量化增加计算开销）
# Q4_K_M 通常比 Q2_K 更快且质量更好

# 4. 减少并发竞争
export OLLAMA_NUM_PARALLEL=1

# 5. 确保模型已预热（已加载到内存）
ollama run llama3.1:8b ""   # 空字符串触发加载
```

7. API 调用超时

现象：长文本生成或大规模模型推理时 HTTP 客户端超时。

解决步骤：

```
# 1. 客户端侧增加超时（以 Python requests 为例）
import requests
requests.post(url, json=payload, timeout=300)  # 5 分钟

# 2. 反向代理侧增加超时（Nginx）
proxy_read_timeout 300s;
proxy_send_timeout 300s;

# 3. 减少生成 Token 数
# API 请求中加入 "options": {"num_predict": 512}

# 4. 使用流式输出避免单次响应过大
"stream": true
```

8. 中文输出乱码

现象：模型输出中文显示为问号或乱码。

解决步骤：

```
# 1. 确认终端编码为 UTF-8
 echo $LANG
# 应为 en_US.UTF-8 或 zh_CN.UTF-8

# 2. 使用支持 UTF-8 的终端（iTerm2、Windows Terminal、VS Code 终端）

# 3. API 调用时确保 HTTP 客户端正确解码 JSON
```

9. Windows WSL2 无法连接宿主机 Ollama

现象：WSL2 内应用无法通过 `localhost:11434` 连接 Windows 宿主机上运行的 Ollama。

解决步骤：

```
# WSL2 中应使用宿主机的 WSL 虚拟 IP，而非 localhost
export OLLAMA_HOST=0.0.0.0  # Windows 侧设置，允许外部连接
# WSL2 侧连接地址改为 Windows 宿主机 IP
curl http://$(cat /etc/resolv.conf | grep nameserver | awk '{print $2}'):11434/api/tags
```

10. 模型回答质量差 / 不符合预期

现象：模型输出与预期偏差大，或频繁产生幻觉。

解决步骤：

```
# 1. 换用更大参数量的模型
# 3B → 7B → 14B → 32B，每档提升显著

# 2. 提高量化精度
# Q4_K_M → Q5_K_M → Q8_0

# 3. 降低 temperature 使输出更确定
# API 或 Modelfile 中设置 temperature 0.1-0.3

# 4. 优化系统提示词（System Prompt）
# 明确约束输出格式、知识边界和拒绝策略

# 5. 使用专用模型替代通用模型
# 代码任务用 Qwen Coder，推理任务用 DeepSeek-R1
```

APPENDIX

附录

A. 各类硬件推荐配置表

A+. Apple Silicon 专用推荐配置

B. 所有环境变量速查表

C. Modelfile 模板合集

D. 学习资源推荐

D+. Ollama 版本升级指南

E. 免责声明

A. 各类硬件推荐配置表

显存 / 内存	推荐模型	推荐量化	适用场景
4 GB	Phi 4 Mini 3.8B	Q4_K_M	轻量对话、边缘设备
6 GB	Llama 3.2 3B / Qwen 3 1.7B	Q5_K_M	笔记本入门、基础问答
8 GB	Llama 3.1 8B / Qwen 3.5 7B	Q4_K_M	开发主力、代码补全
12 GB	Qwen 3.5 14B / DeepSeek R1 14B	Q5_K_M	高性能本地开发
16 GB	Gemma 4 9B / Qwen 3 30B	Q4_K_M	专业推理、多模态
24 GB	Qwen 3.5 32B / Llama 3.3 70B	Q4_K_M	发烧级本地、70B 体验
48 GB	Llama 3.3 70B Q5_K_M	Q5_K_M	高质量大模型生产
64 GB+	Qwen 3.5 122B / DeepSeek V3.2	Q4_K_M	企业级本地部署

A+. Apple Silicon 专用推荐配置

芯片型号	内存	推荐模型配置	预期性能
M1 (8GB)	8 GB	Llama 3.2 3B Q8_0 / Qwen 3 1.7B	15-20 tok/s
M2 (16GB)	16 GB	Llama 3.1 8B Q4_K_M / Qwen 3.5 7B	20-25 tok/s
M3 Pro (36GB)	36 GB	Qwen 3.5 14B Q5_K_M / Gemma 4 9B	18-22 tok/s
M4 Max (64GB)	64 GB	Llama 3.3 70B Q4_K_M / DeepSeek R1 32B	10-15 tok/s
M4 Ultra (128GB)	128 GB	Llama 3.3 70B Q5_K_M / Qwen 3.5 32B Q8_0	15-20 tok/s

> **注意**：Apple Silicon 的统一内存架构意味着 "显存" 即系统内存。上表中的内存指的是芯片可访问的统一内存总量。实际可用模型大小需扣除操作系统和其他应用占用的内存。

B. 所有环境变量速查表

变量名	默认值	说明
`OLLAMA_HOST`	`127.0.0.1:11434`	服务监听地址
`OLLAMA_MODELS`	平台相关	模型存储路径
`OLLAMA_NUM_PARALLEL`	`1`	单模型并发请求数
`OLLAMA_MAX_LOADED_MODELS`	`1`	内存最大驻留模型数
`OLLAMA_KEEP_ALIVE`	`5m`	模型空闲驻留时间
`OLLAMA_NUM_GPU`	自动	GPU 加载层数
`OLLAMA_FLASH_ATTENTION`	`0`	Flash Attention 开关
`OLLAMA_KV_CACHE_TYPE`	`f16`	KV Cache 量化类型
`OLLAMA_DEBUG`	`0`	调试日志开关
`OLLAMA_ORIGINS`	空	CORS 允许来源
`CUDA_VISIBLE_DEVICES`	全部	可见 GPU 设备
`HIP_VISIBLE_DEVICES`	全部	AMD GPU 可见设备

C. Modelfile 模板合集

见第四章 4.4 节，包含编程助手、翻译专家、创意写作、客服机器人四个完整模板。

D. 学习资源推荐

类型	资源	地址
官方文档	Ollama Docs	ollama.com
官方仓库	GitHub	github.com/ollama/ollama
中文社区	Ollama 中文交流	微信公众号、知乎专栏
模型评测	LMSYS Chatbot Arena	lmarena.ai
国内镜像	HF-Mirror	hf-mirror.com
视频教程	B 站 Ollama 部署系列	搜索 "Ollama 本地部署 2026"

D+. Ollama 版本升级指南

```
# macOS / Linux（官方脚本会自动升级）
curl -fsSL https://ollama.com/install.sh | sh

# Homebrew（macOS）
brew upgrade --cask ollama

# Docker
docker pull ollama/ollama:latest
docker-compose up -d

# 升级后验证
ollama --version
```

> **升级提示**：升级 Ollama 不会删除已下载的模型。但建议升级后运行 `ollama list` 确认模型状态正常。若从较早版本（如 v0.18.x 之前）升级，部分模型格式可能不兼容，需重新拉取。

E. 免责声明

本手册仅供技术学习和研究参考。使用 Ollama 运行的大模型可能产生不准确、有偏见或不当的内容，请自行甄别。企业部署前请评估模型许可证（MIT、Apache 2.0、LLaMA License 等）是否符合商用要求。作者不对因使用本手册内容导致的任何直接或间接损失承担责任。

本手册基于 Ollama v0.21.3 编写，最后更新日期：2026年5月。由于开源项目迭代迅速，建议定期查阅官方文档获取最新信息。

关于本手册：本手册由技术社区 contributors 协作编写，遵循 CC BY-SA 4.0 许可证。欢迎通过 GitHub 提交 Issue 或 PR 参与改进。如需商业转载或定制版本，请联系维护团队获取授权。

会员专属资料 · 持续更新

一起玩转 AI 的地方

「河马的CC商业实战」星球里，每周都有看得懂、用得上的 AI 实战：

新栏目「河马带你逛GitHub」每周一期，实战长文、会员资料持续更新。

微信扫码 · 加入「河马的CC商业实战」

河马的CC商业实战 出品