2026年4月版

Gemma 4
完全指南

从入门到本地部署

The Complete Guide to Gemma 4: From Understanding to Local Deployment

---

涵盖型号： E2B · E4B · 26B-A4B · 31B

架构： Dense + MoE · 多模态 · 256K上下文

许可证： Apache 2.0

信息来源： Google官方文档 · Hugging Face · 本地实测

文档版本： v1.0.0

发布时间： 2026-04-03 (build #0)

花叔

公众号「花叔」· B站「AI进化论-花生」

> 知识星球「AI编程：从入门到精通」专属内容

本手册基于Google官方博客、Model Card、Hugging Face发布文档及本地实测编写。所有技术细节以2026年4月最新资料为准。AI工具迭代极快，请结合官方文档验证。

目录

CONTENTS

§01 从Gemma 1到Gemma 4

The Gemma Timeline: From Benchmark Project to Product Line

Google做开源LLM这件事，走了两年多弯路。Gemma 4不是凭空冒出来的，它是前面四代试错的结果。看完这条时间线，你就知道Gemma 4为什么长这样了。

2024年2月：Gemma 1，Google的第一次试水

2024年2月21日，Google发布了Gemma 1，提供2B和7B两个尺寸。

这是Google第一次认真做开源LLM。在此之前，开源这件事基本是Meta在主导（Llama系列），Google的态度一直是「我有Gemini，为什么要开源」。

Gemma 1的定位很清楚：技术验证。它想证明Google也能做出参数效率高的小模型。性能确实不错，在同尺寸模型里排名靠前。但说实话，它更像一个benchmark项目，不是一个你会真正拿来干活的工具。

两个尺寸，纯文本，没有多模态，上下文也不长。社区的反应是：嗯，还行，但我为什么不用Llama？

2024年6月：Gemma 2，性能跳了一大步

四个月后，2024年6月27日，Gemma 2来了。首发两档：9B和27B，7月31日又追加了2B。

27B是Gemma 2真正的亮点。在当时的开源模型里，27B这个尺寸段几乎没有竞品。Llama 3的最小版本是8B，往上就是70B，中间存在一个巨大的空白。Gemma 2 27B刚好插进去。

性能提升也确实明显。在多个benchmark上，Gemma 2 27B的表现接近甚至超过了一些70B级别的模型。Google在技术报告里反复强调的一个词是「知识密度」。

但Gemma 2依然只支持纯文本。没有图片理解，没有长上下文，没有工具调用。它更强了，但它的能力边界没有扩展。

2025年3月：Gemma 3，开始像个产品了

2025年3月10日，Gemma 3发布。这一代的变化是质的。

四个尺寸：1B、4B、12B、27B。更重要的是，**Gemma 3第一次加入了多模态**，支持图片和视频帧输入。上下文窗口从8K一步跳到128K。

我觉得Gemma 3是一个转折点。它不再只是一个在benchmark上跑分的模型，而是开始具备真实场景需要的能力：你可以给它看截图、读文档、处理长文本。

不过在中文圈，Gemma 3的存在感依然有限。同期的Qwen 2.5已经在中文任务上建立了很强的用户心智，DeepSeek也在快速崛起。大部分中文开发者的态度是：Gemma不错，但我的场景里，国产模型更顺手。

2025年6月：Gemma 3n，端侧的新思路

2025年6月26日，Gemma 3n作为一个特殊分支出现，提供E2B和E4B两个型号。

这里的「E」代表「Effective」，也就是有效参数。E2B的意思是有效参数约2B，但总参数更多。**这是PLE（Per-Layer Embeddings）架构第一次出现在Gemma系列里。**

PLE的核心想法是：不同层可以有不同的embedding，让小模型在不增加推理成本的前提下获得更强的表示能力。这个技术后来被Gemma 4继承了。

Gemma 3n还加入了音频输入支持，这在Gemma系列里是第一次。

那些特殊变体

在主线之外，Google还发布过一些专门用途的Gemma变体，快速过一遍：

变体	发布时间	定位
CodeGemma	2024年4月	代码生成和补全，基于Gemma 1微调
PaliGemma	2024年5月	视觉语言模型，基于SigLIP + Gemma
RecurrentGemma	2024年4月	用Griffin架构替代Transformer，实验性质
DataGemma	2024年9月	结合Google的Data Commons，减少幻觉
ShieldGemma	2024年7月	安全过滤器，用于内容审核

这些变体大多没有成为主流，但Google一直在用Gemma做各种方向的探索。Gemma 4能做到今天这样，跟这些技术储备多少有些关系。

2026年4月2日：Gemma 4，真正的产品线

然后就是Gemma 4。

四个型号：E2B、E4B、26B-A4B、31B。覆盖从手机到工作站的全场景。多模态（图片、视频、音频）、256K上下文、MoE架构、工具调用，该有的都有了。

而且这次，许可证从Gemma License换成了Apache 2.0。

这不是一个小变化。之前的Gemma License虽然允许商用，但有不少限制条款，比如模型输出不能用来训练其他模型。Apache 2.0几乎没有这类限制。对于企业用户和开源社区来说，这意味着Gemma 4可以被自由地集成、修改和再分发。

我的判断是：**Google终于把Gemma从一个benchmark项目做成了能当天落地的产品线。**不是说它完美了，但它确实是第一次把性能、多模态、长上下文、本地友好、开放许可这几件事拧到了一起。

以前的Gemma，每一代都有亮点，但总差点什么。Gemma 4是第一个让我觉得「这东西真的可以拿来用」的版本。

> **时间线速查：**Gemma 1（2024.02）→ Gemma 2（2024.06）→ Gemma 3（2025.03）→ Gemma 3n（2025.06）→ Gemma 4（2026.04）。两年四代，Google的迭代节奏比想象中快。

§02 Gemma 4全家福

The Gemma 4 Family: Four Models, One Philosophy

Gemma 4一口气推了四个型号。乍一看有点多，但Google的逻辑其实挺清楚：不同硬件条件，给不同的答案。这一章把四个型号拆开讲，帮你找到最适合自己的那个。

四兄弟，一张表看清楚

型号	有效参数	总参数	上下文	架构	音频
E2B	2.3B	5.1B	128K	Dense	支持
E4B	4.5B	8B	128K	Dense	支持
26B-A4B	3.8B active	25.2B	256K	MoE	不支持
31B	31B	31B	256K	Dense	不支持

几个点拎出来说一下。

E2B和E4B的「E」是Effective的缩写，表示有效参数。它们的总参数比有效参数大，这是PLE架构的特点，后面会解释。两个小模型都支持音频输入，这是从Gemma 3n继承来的能力。

26B-A4B是这次最有意思的型号。总参数25.2B，但每个token只激活3.8B参数。用不到4B的推理成本，跑出接近30B模型的效果。

31B则是纯粹的Dense大模型，不做任何花活，参数全开。适合对性能要求最高、硬件条件也最好的场景。

Dense和MoE到底啥区别

这是理解Gemma 4产品线的核心概念，用一个比喻来说：

Dense模型像一个全能选手。每次接到任务，不管是写代码还是翻译，所有参数都会被激活，全部参与运算。31B Dense就是一个31B参数全上的选手。

MoE模型像一个专家团队。团队里有128个专家（加1个共享专家），但每次处理一个token，只派8个最合适的专家出场。其他120个专家在休息。所以26B-A4B虽然总共有25.2B参数，但每次推理只用3.8B。

推荐

MoE的优势：推理快、显存占用低（相对于同性能的Dense模型）、适合资源受限的环境

不推荐

MoE的代价：模型文件仍然很大（因为128个专家都要加载）、训练更复杂、某些任务上可能不如同参数量的Dense模型

对本地部署来说，MoE是个好消息。因为你关心的是推理速度和生成质量，而不是训练成本。26B-A4B的模型文件虽然比E4B大，但推理时实际计算量只比E4B大一点，效果却好得多。

多模态能力：谁能看、谁能听

Gemma 4的四个型号，多模态能力不完全一样：

能力	E2B	E4B	26B-A4B	31B
文本	支持	支持	支持	支持
图片	支持	支持	支持	支持
视频	支持	支持	支持	支持
音频	支持	支持	不支持	不支持

四个型号都支持图片和视频理解，这是Gemma 4的基本功。但音频输入只有E2B和E4B支持。

其实挺合理的。E2B和E4B定位端侧设备，手机上最常见的多模态输入就是拍照和语音。26B和31B面向开发者工作站，代码、文档、截图是主要输入，音频场景相对少。

架构上有什么新花样

Gemma 4的架构继承了Gemma 3和3n的技术积累，做了几处关键改进。不需要完全理解原理，但知道这几个点能帮你判断它在不同场景下的表现。

PLE：小模型的秘密武器

Per-Layer Embeddings是Gemma 3n引入的技术，在Gemma 4的E2B和E4B上继续使用。

传统模型的所有层共享同一套embedding。PLE让不同层可以有不同的embedding表示，**等于用更多的参数存储知识，但不增加推理时的计算量**。这就是为什么E2B的有效参数是2.3B，但总参数是5.1B。多出来的参数不是浪费，而是让每一层都能更精准地理解输入。

Shared KV Cache：长上下文的效率技巧

处理长文本时，KV Cache（键值缓存）会占用大量显存。Gemma 4在部分层之间共享KV Cache，减少显存开销。

这是26B-A4B能做到256K上下文的关键技术之一。如果每层都独立存KV Cache，256K上下文的显存需求会大到普通硬件根本跑不动。

可变宽高比视觉编码

很多视觉模型处理图片的方式是先resize成正方形，比如224x224。这意味着一张宽屏截图会被压扁，一张长图会被裁切。

Gemma 4不做这种强制resize。它的视觉编码器支持可变宽高比，能按图片的原始比例处理。对于OCR、截图理解、文档解析这些场景，这个细节很重要。你不会因为图片被裁切而丢失关键信息。

Vision Token Budget：70到1120档可配

每张图片会被转换成一定数量的token送给模型处理。Gemma 4允许你配置这个数量，从最少70个token到最多1120个token。

这给了开发者一个很实用的旋钮：处理简单的图标或按钮截图，用70个token就够了，省时间；处理复杂的文档页面或UI截图，调到1120个token获得更高精度。

26B-A4B到底强在哪

架构说够了，看实际表现。

Google在LMArena上公布了Gemma 4的成绩。LMArena是目前社区公认最有参考价值的模型评测平台之一，采用人类偏好投票的方式排名，比纯benchmark更接近真实体验。

模型	LMArena分数	Active参数	备注
Gemma 4 31B	~1452	31B	Dense，全参数推理
Gemma 4 26B-A4B	~1441	3.8B	MoE，只激活4B参数

请注意这两个数字的差距：**只有11分。**

31B动用了全部31B参数，26B-A4B只用了3.8B。推理成本差了近8倍，但性能几乎持平。这就是MoE架构的威力。

放到更大的格局里看：

Gemma 4 31B和26B-A4B的LMArena分数，已经接近GLM-5和Kimi K2.5的水平。但后两者的参数量大得多。**用不到30B的参数打出同级别表现，这就是Google说的intelligence-per-parameter。**

和其他开源模型比一比

vs Llama 4 Scout

Meta的Llama 4 Scout是最直接的竞争对手。Scout的上下文窗口达到10M tokens，这一点Gemma 4的256K完全没法比。如果你的场景需要处理超长文档或完整代码库，Scout有明显优势。

但Scout的总参数109B（17B激活参数，16个专家）。虽然是MoE架构只激活一部分，这个模型体积对本地部署来说还是比Gemma 4大得多。**Gemma 4 26B-A4B的甜点在于：普通笔记本电脑就能跑，不需要GPU服务器。**

vs Qwen 3

Qwen 3在数学和代码任务上的表现可能更强，这是阿里长期投入的优势领域。中文理解和生成方面，Qwen作为国产模型也有天然优势。

但Gemma 4在多模态能力和端侧优化上更出色。如果你需要一个能同时处理图片、文档、OCR的本地模型，Gemma 4的选择面更宽。而且Apache 2.0的许可证比Qwen的更开放。

选哪个型号？看你的硬件

OK，落到选型上。

> **核心建议**
>
> **选型建议（按硬件条件）：**
>
> **手机或树莓派** → E2B。有效参数只有2.3B，4-bit量化后4GB内存就能跑。支持音频输入，适合做端侧语音助手。
>
> **普通笔记本（8-16GB内存）** → E4B。4.5B有效参数，量化后8GB左右内存可用。能力比E2B强不少，仍然支持音频。
>
> **高配笔记本或工作站（24GB+内存）** → 26B-A4B。这是甜点位。4-bit量化后大约需要16-18GB内存，24GB的Mac可以比较舒服地跑。256K上下文，MoE架构，性价比最高的本地模型之一。
>
> **32GB+内存且追求极致** → 31B。全Dense架构，不做参数节省，给你最强的单模型性能。代价是推理速度比26B-A4B慢，显存占用也更大。

我个人的推荐是**26B-A4B**。

原因很简单：性能和31B几乎一样，推理成本低得多。大多数本地场景下，你不需要为了11分的LMArena差距去承受8倍的推理开销。

Google官方博客里有一句话说得挺精准的：Gemma 4追求的是byte for byte, the most capable open models。翻译成人话就是：每一块显存都不浪费。26B-A4B是这个理念最极致的体现。

快速决策：如果你只想记住一个型号，记26B-A4B。它大概率是2026年上半年本地部署的最佳选择之一。

§03 你的电脑能不能跑

Can Your Machine Handle It?

不管Gemma 4的benchmark有多好看，你电脑跑不动就白搭。这一章帮你快速判断：手上的硬件，到底适合跑哪个型号。

先搞懂一件事：量化

大模型的原始权重是用16位浮点数（FP16）存的，精度高但占空间大。**量化就是把这些数字的精度降低，换取更小的体积和更低的显存占用。**

打个比方：原始模型是无损FLAC音乐文件，量化后的模型是MP3。体积小了一大截，但听感上大部分人分不出区别。当然，压缩率越高，失真越明显。

你会在模型名字里看到这些后缀：

量化等级	含义	质量	体积
FP16	原始精度，不压缩	最高	最大
Q8_0	8位量化	接近原版	约FP16的一半
Q5_K_M	5位量化	很好	更小一些
Q4_K_M	4位量化	够用	约FP16的四分之一

还有一种特殊的量化方式叫**QAT（Quantization Aware Training）**，Google官方出品。它不是训练完再压缩，而是训练的时候就考虑了量化的影响，所以同等压缩率下质量最好。如果你看到模型名带QAT后缀，优先选它。

硬件需求对照表

这张表是你做决定的核心依据。找到你的内存/显存，横向对比能跑什么：

型号	FP16（原始）	Q8（8位）	Q4（4位）	Mac最低配置	NVIDIA最低配置
E2B	~10GB	~7GB	~4GB	任意Mac	任意独显
E4B	~16GB	~10GB	~6GB	M系列 16GB	RTX 3060 12GB
26B-A4B	~50GB	~28GB	~17-18GB	M系列 24GB	RTX 4090 24GB
31B	~62GB	~32GB	~19-20GB	M系列 32GB	RTX 4090 24GB（紧）

> **核心建议**
> 对大多数人来说，**Q4_K_M是最佳平衡点**。体积够小、质量够用。如果显存宽裕（比如还剩好几个GB），可以试试Q5_K_M。显存完全够的话，Q8_0接近原版效果。

MoE的显存陷阱

这是很多人会搞错的地方。

26B-A4B是MoE（Mixture of Experts）架构，每次推理只激活3.8B参数，推理速度接近4B模型。听起来很美好。但问题是：所有25.2B参数必须全部加载到显存里，一个都不能少。

为什么？因为模型不知道下一个token需要用到哪些专家。每次推理只激活其中一部分，但选哪些取决于输入内容。所以所有专家都得在显存里待命。

> **注意**
> **不能只加载3.8B参数。**MoE的「激活参数少」是指计算量小、推理快，不是指占用内存少。26B-A4B的显存需求看的是总参数量25.2B，不是激活参数3.8B。

这也解释了为什么26B-A4B的Q4量化版需要约18GB显存。如果你只看「激活3.8B」就以为8GB够了，会直接报内存不足。

Mac用户特别说明

如果你用的是Apple Silicon Mac（M1/M2/M3/M4），有一个天然优势：**统一内存**。CPU和GPU共享同一块内存，不用像PC那样纠结「显存多大」的问题。你的Mac有多少GB内存，就有多少GB可以给模型用（当然系统和其他应用也要占一部分）。

各代芯片跑模型差异不大，M1和M4的推理速度有差距，但不是决定性的。**真正决定你能跑什么的是内存总量**，不是芯片代数。

实测数据：我在M4 Pro 24GB上跑26B-A4B的Q4版本，可以正常运行。24GB里扣掉系统占用，剩下的空间刚好够把模型加载进去。不算宽裕，但能用。

如果你是8GB的MacBook Air，别为难自己了，E2B是你的菜。16GB可以试E4B。24GB是26B-A4B的起步线。32GB以上就比较自由了，31B的Q4或者26B-A4B的Q8都能考虑。

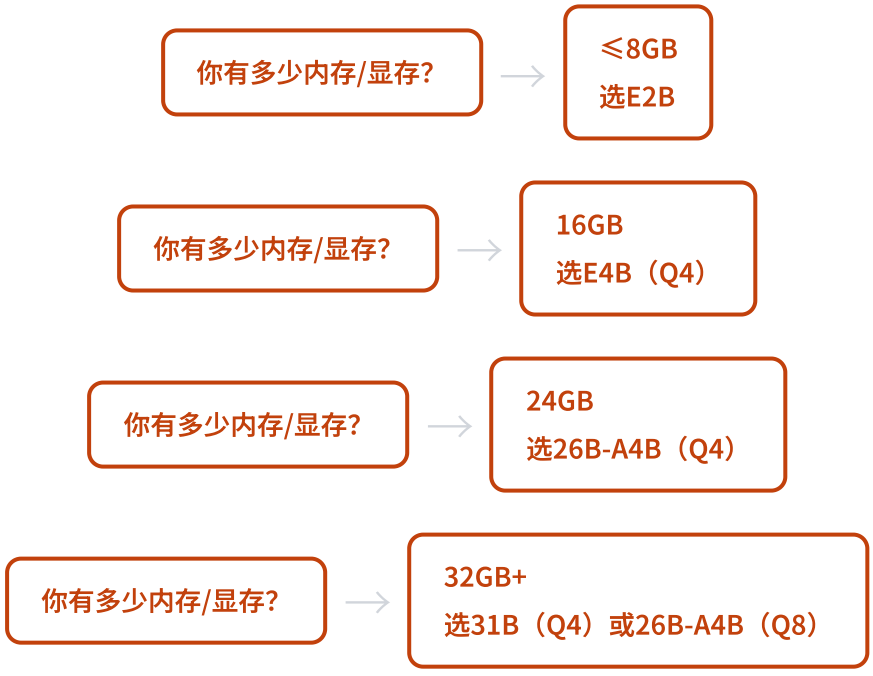
量化版本怎么选

确定了型号，还得选量化版本。原则很简单：

量化版本	推荐场景
QAT（如果有）	Google官方量化，同等大小下质量最好，优先选
Q4_K_M	推荐大多数人，平衡点
Q5_K_M	显存稍有富余时选，质量比Q4好一点
Q8_0	显存充足时选，几乎无损

不确定选哪个？Q4_K_M。不会错。

快速决策流程



一句话建议：不要追求跑最大的模型。在你的硬件条件下，选一个能流畅运行的版本，比强行塞一个卡顿的大模型体验好得多。

§04 Ollama部署实战

Deploy with Ollama: Step by Step

Ollama是目前本地跑大模型最省事的工具，「大模型的Docker」：一条命令拉模型，一条命令跑起来。这一章从安装到API调用，手把手走一遍。

安装Ollama

三个平台都支持，挑你的系统看。

1. **macOS**

   推荐用Homebrew安装，也可以从官网下载安装包：

   ```
   # 方式一：Homebrew（推荐）
   brew install ollama

   # 方式二：官方安装脚本
   curl -fsSL https://ollama.com/install.sh | sh
   ```

2. **Windows**

   用winget一行搞定：

   ```
   winget install ollama
   ```

   也可以去 `ollama.com` 下载安装包，双击安装。

3. **Linux**

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

安装完成后，终端输入 `ollama --version` 确认安装成功。

拉取Gemma 4模型

Ollama的用法和Docker很像：先pull（下载），再run（运行）。根据你在§03选好的型号，执行对应的命令：

```
# 最小的，任何机器都能跑
ollama pull gemma4:e2b

# 轻量多模态，16GB内存起步
ollama pull gemma4:e4b

# 甜点位，24GB内存起步（推荐）
ollama pull gemma4:26b

# 满血版，32GB内存起步
ollama pull gemma4:31b
```

各型号在Ollama上的实际文件大小：

型号	Ollama标签	下载大小
E2B	gemma4:e2b	7.2GB
E4B	gemma4:e4b	9.6GB
26B-A4B	gemma4:26b	18GB
31B	gemma4:31b	20GB

下载速度看网络，26B-A4B的18GB可能得等一会儿。

> **核心建议**
>
> **Gemma 4在Ollama上是Day-1支持的**，发布当天就能pull。Google和Ollama这次的配合比以前好很多，不用等社区打包了。

第一次对话

模型下载完，一条命令就能开始聊：

```
ollama run gemma4:26b
```

等几秒钟加载模型，出现 `>>>` 提示符后就可以打字了。输入问题，回车，等它生成回复。按 `Ctrl+D` 退出。

第一次加载会比较慢，因为整个模型要从磁盘读进内存。后续再跑会快很多，Ollama会把模型保持在内存里一段时间。

多模态：喂图片

Gemma 4支持图像输入。你可以把图片直接丢给它，让它描述、分析、提取文字：

```
# 描述一张图片
ollama run gemma4:26b "请描述这张图片" --images photo.jpg

# OCR：从截图中提取文字
ollama run gemma4:26b "提取图中的所有文字" --images screenshot.png

# 分析UI设计
ollama run gemma4:26b "这个界面的设计有什么问题？" --images app-ui.png
```

图片路径可以是相对路径或绝对路径。如果图片不在当前目录，用完整路径就行。

API Server：给其他工具用

Ollama不只是个聊天工具，它同时是一个API服务器。**启动Ollama后，本地就自动开了一个API端口（11434），**任何支持HTTP的工具都能接入。

后面章节要讲的OpenClaw接入、代码编辑器集成，都依赖这个API。

Ollama原生API

```
curl http://localhost:11434/api/chat -d '{
  "model": "gemma4:26b",
  "messages": [
    {"role": "user", "content": "用Python写一个快速排序"}
  ]
}'
```

OpenAI兼容API

很多工具（比如Cursor、Continue、Chatbox）认的是OpenAI格式的API。**Ollama直接兼容这个格式，**不用装任何额外的东西：

```
curl http://localhost:11434/v1/chat/completions \
  -H "Content-Type: application/json" \
  -d '{
    "model": "gemma4:26b",
    "messages": [
      {"role": "user", "content": "你好"}
    ]
  }'
```

在那些工具的设置里，把API地址填 `http://localhost:11434/v1`，模型名填 `gemma4:26b`，API Key随便填一个（Ollama不校验），就能用了。

> **核心建议**
>
> **记住这个地址：** `http://localhost:11434/v1` 。后面所有「接入本地模型」的场景，都会用到它。这是Ollama的OpenAI兼容接口，几乎所有主流工具都认。

常用命令速查

命令	用途
`ollama list`	查看已下载的模型
`ollama ps`	查看正在运行的模型
`ollama pull 模型名`	下载模型
`ollama run 模型名`	运行模型（交互式）
`ollama rm 模型名`	删除模型（释放磁盘空间）
`ollama show 模型名`	查看模型详情

常见问题

注意

pull很慢或超时

国内网络下载Ollama模型可能比较慢。可以考虑：设置代理、使用国内镜像源、或者直接下载GGUF文件手动导入（见 § 05）。

注意

内存不够，模型加载失败

报错信息通常会提示内存不足。解决办法：换一个更小的型号（比如从26B-A4B降到E4B），或者用更低的量化版本（从Q8降到Q4）。关掉其他占内存的应用也能帮上忙。

注意

GPU没被识别

Mac用户一般不会遇到这个问题，Apple Silicon自动走Metal加速。NVIDIA用户需要确认CUDA驱动已安装且版本匹配。终端运行 `nvidia-smi` 看能不能正常显示GPU信息。

注意

模型能跑但速度很慢

多半是模型太大，一部分权重被swap到了磁盘，推理速度断崖式下降。老话：**换小一号的模型或更低的量化版本**，流畅运行比强行塞大模型重要得多。

下一步：如果Ollama暂时不支持Gemma 4，或者你想要更精细地控制模型参数，§05会介绍llama.cpp部署方案。两条路线最终都能跑起来，选你顺手的就好。

§05 LM Studio + llama.cpp部署

Alternative Deployment: GUI and CLI

Ollama不是唯一选择。LM Studio有图形界面，llama.cpp能精确控制每个参数，MLX则让Apple Silicon跑满。

LM Studio：本地版ChatGPT界面

如果你用过ChatGPT的网页版，LM Studio的体感很接近。区别是，它跑在你自己的电脑上，用的是开源模型，不花钱，数据不出本地。

可以理解成一个带图形界面的本地模型运行器。下载模型、加载模型、聊天、调参数，全在GUI里完成，不碰命令行。

去 `lmstudio.ai` 下载对应系统的安装包，Mac、Windows、Linux都有。装完打开就能用。

搜索和下载Gemma 4

打开LM Studio后，在左侧的模型搜索栏搜「gemma-4」，会看到一堆结果。你要关注的是GGUF格式的量化版本。

量化版怎么选？这里给一个速查：

量化格式	大小（26B A4B）	质量	推荐场景
Q3_K_M	~10GB	够用，偶尔有瑕疵	16GB内存的机器
Q4_K_M	~14GB	性价比最高	24GB内存，日常使用首选
Q5_K_M	~16GB	接近原版	32GB+内存，追求质量
Q6_K	~19GB	几乎无损	64GB+内存

多数情况下，Q4_K_M就够了。这是社区公认的性价比甜点。质量损失很小，但省下来的内存可以留给更长的上下文。

GUI聊天和本地API

模型下载完，点加载，就可以直接在LM Studio的对话界面里聊天。温度、top-p、系统提示词这些参数都可以在右侧面板里调。

不过LM Studio更有意思的地方在于：**它内置了一个OpenAI兼容的API Server**。在左侧菜单找到「Local Server」，点启动，API就跑起来了：

```
# LM Studio启动后，API默认在
http://localhost:1234/v1/chat/completions

# 用curl测试
curl http://localhost:1234/v1/chat/completions \
  -H "Content-Type: application/json" \
  -d '{
    "model": "gemma-4-26b-a4b",
    "messages": [{"role": "user", "content": "你好"}]
  }'
```

也就是说，任何支持OpenAI API格式的工具，都能直接对接LM Studio。不用改代码，把API地址换成 `localhost:1234` 就行。

llama.cpp：底层引擎

其实你用过的很多工具，底层都在调llama.cpp。Ollama用它，LM Studio也用它。那些工具帮你包了一层壳，而llama.cpp是直接操作引擎本身。

好处是你能控制一切。代价是你得自己处理一切。

安装

```
# macOS（最简单的方式）
brew install llama.cpp

# Windows
winget install ggml.llamacpp

# 从源码编译（需要CUDA或Metal加速时）
git clone https://github.com/ggml-org/llama.cpp
cd llama.cpp
cmake -B build
cmake --build build --config Release
```

Mac用户直接brew装就行。Windows用winget。如果你有NVIDIA显卡想用CUDA加速，才需要从源码编译，编译时加上 `-DGGML_CUDA=ON` 参数。

启动llama-server

llama.cpp最实用的功能是llama-server。它会启动一个OpenAI兼容的API服务，和Ollama、LM Studio的API格式完全一致。

```
# 最简单的启动方式：直接从HuggingFace拉模型并启动
llama-server -hf ggml-org/gemma-4-26B-A4B-it-GGUF:Q4_K_M

# API在 http://localhost:8080/v1
# 启动后就可以用curl或任何OpenAI兼容客户端调用
```

`-hf` 参数会自动从HuggingFace下载模型，用的是ggml-org的官方量化版本。第一次跑需要下载，之后会缓存在本地。

这也是Hugging Face官方推荐的启动方式。他们在Gemma 4的launch blog里直接给了这行命令，day-0就准备好了本地部署路径。

花叔实测：M4 Pro 24GB跑26B A4B

我在M4 Pro 24GB上用llama.cpp跑了Gemma 4 26B A4B的Q4_K_M量化版。

实际占用内存大约14-15GB，剩下的空间够系统和日常应用。推理速度大概每秒15-20个token，一段200字的中文回复等几秒。不算快，但够用。

体感上，llama.cpp的推理速度和Ollama非常接近，因为底层就是同一个引擎。区别在于你可以更细粒度地控制参数，比如线程数、batch size、GPU层数这些。

MLX：Apple Silicon专属

如果你用的是Mac，还有第三个选择：MLX。这是Apple自己的机器学习框架，专门为M系列芯片优化，直接用Metal加速。

```
# 安装
pip install -U mlx-vlm

# 运行（以E4B为例）
mlx_vlm.generate --model google/gemma-4-E4B-it \
  --prompt "你好"
```

MLX和Apple Silicon的配合更紧密。Ollama和llama.cpp虽然也支持Metal，但MLX是原生的，理论上效率更高。

另外有个叫TurboQuant的技术：3.5-bit的KV cache量化，能进一步压缩运行时的内存占用。对于24GB或更少内存的Mac，这个优化可能决定你能不能把上下文窗口开得更大。

不过MLX的生态不如llama.cpp成熟，工具链也不如Ollama方便。**除非你有明确的性能需求，否则Ollama或llama.cpp已经够用了。**

三种方式怎么选

特性	Ollama	LM Studio	llama.cpp
上手难度	⭐	⭐	⭐⭐⭐
GUI界面	❌	✅	❌
API兼容	OpenAI格式	OpenAI格式	OpenAI格式
量化控制	自动选择	手动选择	完全控制
适合谁	快速跑通	想要GUI的人	极客和开发者
底层引擎	llama.cpp	llama.cpp	就是它本身

挺有意思的一点是：**这三个工具提供的API格式完全一致，都是OpenAI兼容的。**你用哪个启动模型，对上层应用来说没区别。Ollama在 `localhost:11434`，LM Studio在 `localhost:1234`，llama.cpp在 `localhost:8080`，仅此而已。

核心建议

如果你之前没用过本地模型，从Ollama开始。如果你想要图形界面方便切换模型和调参，用LM Studio。如果你需要精确控制推理参数或者做批量处理，用llama.cpp。三者不冲突，可以同时装。

§06 接入OpenClaw和龙虾产品

Connect to OpenClaw and Local AI Agents

模型跑起来只是第一步。接下来，让它真正为你工作。

为什么要接入Agent

模型跑起来了，能聊天，能回答问题。但说实话，拿本地模型当聊天机器人，有点浪费。

聊天这件事，云端模型做得更好。Claude、GPT、Gemini的在线版，对话质量还是比本地模型强不少。

本地模型真正有意义的用法是：当你自己的私有AI后端。

比如处理本地文档、辅助写代码、分析数据、自动化工作流。这些任务交给一个本地跑的Agent，不花钱，不传数据，随时可用。

这就是OpenClaw和其他龙虾产品要做的事。

OpenClaw是什么

OpenClaw是一个开源的AI Agent框架。逻辑很直接：

你给它一个OpenAI兼容的API地址 → 它通过这个API调用模型 → 帮你执行任务。

把本地Ollama或llama.cpp跑的Gemma 4接进去，你就有了一个免费的、私有的、跑在自己电脑上的AI Agent。模型开源，框架开源，整条链路零API费用。

三条接入路径

不管你用哪种方式跑模型，接法都一样：告诉OpenClaw API在哪、用的是什么模型。

路径1：Ollama作为后端

```
# Ollama默认提供OpenAI兼容API
# 启动Ollama后，API自动可用

# OpenClaw配置：
# base_url: http://localhost:11434/v1
# model: gemma4:26b-a4b
```

这是最省事的方式。Ollama装好、模型拉好，API就在那里，不需要额外操作。

路径2：llama.cpp作为后端

```
# 先启动llama-server
llama-server -hf ggml-org/gemma-4-26B-A4B-it-GGUF:Q4_K_M

# OpenClaw配置：
# base_url: http://localhost:8080/v1
# model: 不需要指定（llama-server同时只跑一个模型）
```

llama.cpp的好处是启动参数可以精确控制。如果你对推理速度有要求，或者想指定GPU层数、线程数，这条路更灵活。

路径3：LM Studio作为后端

```
# 在LM Studio中加载模型
# 启动Local Server

# OpenClaw配置：
# base_url: http://localhost:1234/v1
# model: 在LM Studio中加载的模型名
```

适合喜欢GUI的人。在LM Studio里切换模型、调参数，同时让OpenClaw连着API端口，两边各干各的。

在OpenClaw中配置

配置的核心就一件事：把API endpoint指向本地。

打开OpenClaw的Settings，找到模型配置，把API地址从云端改成本地。如果你用Ollama，就填 `http://localhost:11434/v1`；用llama.cpp就填 `http://localhost:8080/v1`。

配置文件位置在 `~/.openclaw/openclaw.json`，你也可以直接编辑这个JSON文件。关键字段是 `base_url` 和 `model`。

```
// openclaw.json 中的模型配置示例
{
  "providers": {
    "local-gemma": {
      "base_url": "http://localhost:11434/v1",
      "model": "gemma4:26b-a4b",
      "api_key": "not-needed"
    }
  }
}
```

注意 `api_key` 这个字段。本地模型不需要真的API key，但有些客户端要求这个字段存在，随便填一个字符串就行。

不只是OpenClaw

前面反复提到，Ollama、LM Studio、llama.cpp提供的都是标准的OpenAI兼容API。所以**任何支持这个格式的工具都能接入你的本地Gemma 4**：

工具	类型	接入方式
OpenClaw	AI Agent框架	Settings中修改API endpoint
Hermes	本地AI助手	配置OpenAI兼容API
Open Code	AI编程助手	设置base_url指向本地
Continue.dev	IDE插件	config.json中添加本地模型
任何OpenAI SDK应用	自定义	修改base_url即可

Hugging Face在Gemma 4的发布博客里也专门提到了这一点。他们说：

> **官方原文：**We worked on making sure the new models work locally with agents like openclaw, hermes, pi, and open code.

这不是社区自己折腾出来的，是Google和Hugging Face有意为之。发布当天，本地Agent的接入路径就已经铺好了。

一个完整的工作流

把上面这些串起来，一个典型的工作流长这样：

处理文档、写代码、分析数据、整理文件。这些任务不需要云端最强的能力，但需要隐私、需要免费、需要随时可用。Gemma 4 26B A4B在这些场景下够用了。

举个例子：你有一批本地PDF需要提取关键信息整理成表格。以前你可能会传到ChatGPT或Claude，现在直接让本地的OpenClaw去做。文件不离开你的电脑，不花API费用，而且Gemma 4本身就支持文档理解和OCR。

我的判断

这才是本地模型该有的用法。不是拿它和Claude比谁聊天更聪明，是把它当成你自己的私有AI后端。

免费、本地、不传数据、随时可用。一个26B级别的多模态模型满足这些条件时，它的价值就不是跑分能衡量的了。

你不需要它是最聪明的模型。你需要它是你自己的。

> **核心建议**
> 如果你已经在用OpenClaw或其他龙虾产品，只需要改一个API地址就能接入本地Gemma 4。如果你还没用过，这可能是一个不错的开始理由。

§07 端侧模型的黄金场景

Where Local Models Truly Shine

本地模型不是云端AI的平替。它有自己的领地，有些事只有它能干。

先说最没争议的：隐私文档

发票、合同、体检报告、银行流水、公司会议纪要。

这些东西的共同点是：你不想让它们离开你的电脑。哪怕对方是OpenAI、Google、Anthropic。不是信不过他们的安全措施，是这类数据一旦上传，你就失去了物理控制。合规部门不在意云端隐私政策写得多漂亮，他们只认一件事：数据有没有出过内网。

本地模型把这个问题直接消灭了：数据从来没离开过你的硬盘。

Gemma 4刚好踩中了这个场景。Google官方model card里把Document/PDF parsing、OCR、handwriting recognition都列为主打能力。实测拍一张发票照片喂给26B A4B，金额、日期、税号这些关键字段都能准确提取。E4B也行，但26B的准确率更高，特别是手写字迹和印章重叠的区域。

律师事务所、会计事务所、或者任何经手大量敏感文档的团队，这可能是本地模型最直接的使用理由。

接下来聊代码

代码补全、解释一段看不懂的函数、重构一块写得丑的逻辑、修一个小bug。这些事每天都在做，每次都调云端API的话，一个月下来费用不少，代码也传出去了。

本地代码助手的好处很直接：**不泄露代码、不花钱、离线也能用。**飞机上、高铁上、公司内网里，笔记本在手边就行。

前面几章讲过，Ollama、LM Studio、llama.cpp都提供OpenAI兼容API。VS Code的Continue插件、JetBrains的AI Assistant，或者任何支持自定义API endpoint的编辑器插件，都能直接指向 `localhost`。

```
# Continue插件配置示例（~/.continue/config.json）
{
  "models": [{
    "title": "Gemma 4 26B Local",
    "provider": "ollama",
    "model": "gemma4:26b-a4b"
  }]
}
```

26B A4B在coding benchmark上表现不差。Google的model card报了LiveCodeBench v6得分77.1%，Codeforces ELO 1718。和云端顶级模型比有差距，但作为一个本地免费的代码助手，已经越过了「能用」的线。

日常代码补全和解释，它处理得很稳。复杂的跨文件重构，还是交给云端靠谱。

文档处理和代码之外，还有一个我觉得很实用的场景：本地知识问答

你有一堆自己的文档。产品手册、会议记录、读书笔记、技术文档。你想随时问它们问题，但不想搭复杂的搜索系统，也不想把文件传到云端。

本地RAG（Retrieval-Augmented Generation）就是干这个的。文档切片、建向量索引、检索相关片段、喂给本地模型生成回答。整个流程在你电脑上闭环。

Gemma 4在这个场景有一个很大的优势：256K上下文窗口。这意味着你甚至可以跳过RAG的复杂管线，直接把一份很长的文档整个塞进上下文里让它回答问题。一本10万字的技术手册，大概占6-8万token，塞得进去。

当然上下文越长，推理越慢，内存占用也越大。实际操作中建议控制在8K-32K，既能覆盖多数文档片段，又不至于卡死机器。

对于每天需要反复查阅内部文档的人，这比每次都打开浏览器搜效率高得多。速度稳定，成本为零，不受网络影响。

说到多模态，这可能是Gemma 4最被低估的能力

它不只处理文字。图片、视频帧、音频都能吃。

拍一张产品标签的照片，提取营养成分表。截一张报错界面的图，分析问题在哪。录一段语音备忘，转成文字摘要。截一张App界面，描述UI元素和布局结构。

其中音频输入是E2B和E4B支持的能力，26B A4B目前聚焦在文本和图像。**如果你需要音频理解，小模型反而是更合适的选择。**

视频分析的做法是抽帧。按固定间隔截取画面，每一帧当图片理解。不算优雅，但能用。监控画面分析、教学视频内容提取、产品演示录屏的自动标注，都可以走这条路。

Google在model card里专门提到了screen and UI understanding。**对开发者特别有用**：截一张设计稿或竞品界面的图，让模型分析布局、提取配色、甚至生成还原代码。

再往前走一步：让本地模型自己干活

Agent是2026年AI圈最热的词之一。不过多数人聊Agent，默认在说云端API。

其实有一类Agent工作流，本地模型反而更合适：**固定流程、固定工具集、重复执行。**

比如每天早上自动整理邮件摘要。比如定时从几个数据源拉数据、生成日报。比如监控一个文件夹，有新文件进来就自动分析内容、打标签、归档。

这类任务模式固定、频率高。每次都调云端API的话，一天跑几百次，cost很快上来。而且还容易撞rate limit。

Gemma 4支持Function Calling，Google model card里直接把它列为核心能力。Hugging Face的launch blog也提到了和OpenClaw、Hermes这类本地Agent框架的day-0集成。

本地Agent没有API费用，没有rate limit，没有网络延迟。可以全天候跑着，不心疼账单。代价是你的电脑得一直开着，推理期间CPU/GPU占用比较高。

把视野再拉远一点：手机和嵌入式设备

前面说的都是笔记本和台式机。但Gemma 4的野心不止于此。

Google官方给的数据挺有说服力：E2B在树莓派5上跑出了133 tok/s的预填充速度和7.6 tok/s的生成速度，内存占用不到1.5GB。

7.6 tok/s不算快，但嵌入式场景够了。想想看：一个树莓派大小的设备，不联网，就能做中文问答、文档分析、简单的图片理解。智能家居网关、工厂车间的边缘服务器、没有网络覆盖的野外观测站，都能派上用场。

Android端更方便。**Google直接提供了AI Edge Gallery，手机上装个App就能跑Gemma 4。**不用Root，不用配环境，不用命令行。对普通用户来说，这可能是门槛最低的体验方式。

E2B是嵌入式和移动端的主力选手。E4B在手机上也能跑，但会更吃内存。26B A4B就别想了，那是给笔记本和工作站准备的。

最后一个场景有点不一样：学习

如果你想理解大模型是怎么工作的，最好的方式不是读论文，是自己跑一个。

本地模型让你可以做很多云端API做不了的事：

调temperature从0到2，一句一句看输出怎么变化。改system prompt，观察模型行为的变化。看每一层的attention pattern。用LoRA做小规模微调，体会训练的过程。把模型的回答和Claude、GPT的回答做逐句对比，理解不同模型的差异。

Apache 2.0协议意味着你可以随便改、随便部署、做商业项目，不用和Google签任何额外协议。基于Gemma 4做微调、蒸馏、二次开发，然后把成果卖钱，完全合法。对开发者和创业团队来说，这个自由度很大。

对学生和研究者来说，这是一个免费的、开源的、性能不错的实验平台。比读十篇综述更能帮你建立直觉。

> **花叔的判断：**本地模型不是要替代云端AI，是你的私人后备力量。有些事不值得每次花钱调API，有些数据你不想传出去，有些场景就是需要离线能用。Gemma 4把这几件事同时做到了不错的水平。它不是最强的，但可能是你用得最顺手的。

§08 进阶玩法与调优

Advanced Tips and Tuning

模型跑起来只是60分。这一节帮你调到80分。

System Prompt定制

通用模型的默认行为是什么都能聊一点，但什么都不够专。给它一个明确的角色定义，输出质量会明显不一样。

在Ollama里，方法是写一个Modelfile：

```
# 文件名：Modelfile
FROM gemma4:26b-a4b

SYSTEM "你是一个专业的代码审查助手。请用中文回复。审查时重点关注：安全漏洞、性能问题、代码可读性。给出具体的改

PARAMETER temperature 0.3
PARAMETER num_ctx 8192
```

然后用两行命令创建和运行你的自定义模型：

```
# 基于Modelfile创建自定义模型
ollama create my-code-reviewer -f Modelfile

# 使用自定义模型
ollama run my-code-reviewer
```

这个自定义模型会出现在你的Ollama模型列表里，和原版并列。你可以创建好几个不同的变体：一个做代码审查的、一个做文档翻译的、一个做数据分析的。每个都有针对性的system prompt和参数配置。

在llama.cpp和LM Studio里也可以设system prompt，只是方式不同。llama.cpp在启动server时通过 `--system-prompt-file` 参数从文件加载，或者在API请求中动态传入。LM Studio在GUI的对话设置里直接填。效果一样。

参数调优

跑模型不调参数，就像买了台相机永远用自动挡。能拍，但出不了好片。

temperature

控制输出的随机性。数字越低，回答越确定、越保守；越高，越有创意、越可能跑偏。

场景	推荐值	原因
代码生成/修bug	0.1 - 0.3	要准确，不要惊喜
文档摘要/翻译	0.3 - 0.5	稍有灵活性但不偏离原意
创意写作/头脑风暴	0.7 - 1.0	需要多样性和意外感
角色扮演/故事创作	0.8 - 1.2	越不可预测越有趣

top_p

和temperature配合使用，控制模型从多大范围的词汇里选下一个token。0.9是多数场景的默认值，一般不用动。输出太单调就提到0.95，太散漫就降到0.8。

num_ctx（上下文长度）

直接影响内存占用和推理速度。

Ollama默认值是2048，对多数对话来说太短了。建议至少设到4096，日常用8192比较舒服。26B A4B理论上支持256K上下文，但24GB内存的机器上开这么大，内存直接爆。

```
# Ollama在对话中设置上下文长度
# 启动后输入：/set parameter num_ctx 8192

# 或者在Modelfile里固定（推荐）
PARAMETER num_ctx 8192
```

经验法则：**上下文每增加1倍，内存额外占用大约增加10-15%。**8192对24GB机器来说很安全，16384也能撑住，再往上就需要试了。

repeat_penalty

防止模型车轱辘话来回说。默认1.1够用。模型在重复自己的话，提到1.2-1.3。太高输出会变得生硬，不建议超过1.5。

Vision Token Budget

这是Gemma 4比较独特的功能，调优多模态性能的关键。

给模型传图片时，模型需要把图片转成token来理解。**Vision Token Budget控制的就是分配多少token给这个过程。**token越多，图片被「看」得越细；越少，处理越快、内存占用越低。

一共5档：

Token Budget	适用场景	速度
70	粗略判断图片内容（是猫还是狗）	最快
140	简单描述、分类	快
280	日常图片理解（推荐默认值）	适中
560	OCR文字提取、文档扫描	较慢
1120	分析复杂图表、细粒度UI理解	最慢

日常用280就够了。需要从图片里提取文字（比如拍照识别发票），用560。分析数据密集的图表或复杂的UI截图，上1120。

不需要每张图都用最高档。**多数时候280的效果已经够好，而速度是1120的好几倍。**只有在你觉得模型「看漏了东西」的时候，再把档位调上去。

Extended Thinking模式

Extended Thinking让模型在输出答案之前，先做一轮内部推理。可以理解成让模型「先想清楚再回答」。

数学题、逻辑推理、复杂的代码debug、需要综合多个条件做判断的问题，开了thinking之后准确率会上一个台阶。

代价也直接：输出变长，延迟变大。模型会先输出一段思考过程（thinking tokens），然后才给最终答案。**任务很简单的话，比如翻译一句话或者写个hello world，没必要开。**

```
# 在transformers API中使用
# 通过apply_chat_template启用thinking模式
text = processor.apply_chat_template(
    messages,
    tokenize=False,
    add_generation_prompt=True,
    enable_thinking=True  # 启用推理模式
)
inputs = processor(text=text, return_tensors="pt").to(model.device)
outputs = model.generate(**inputs, max_new_tokens=2048)
```

在Ollama里，thinking模式已经内置在 `gemma4` 系列模型中，不需要额外配置。如果你想关掉它来加快速度，可以在请求中设置 `"think": false`。

GPU Offload优化

推理计算可以跑在CPU上，也可以跑在GPU上。GPU快得多，但显存有限。模型太大GPU放不下的时候，就需要部分卸载：一部分层跑GPU，剩下的跑CPU。

Mac用户

Mac基本不用操心这个。M系列芯片的统一内存架构，CPU和GPU共享同一块内存，系统自动分配。确保内存够就行，剩下的交给macOS。

NVIDIA显卡用户

显存不够放整个模型的话，llama.cpp提供了精细的控制：

```
# -ngl 控制多少层放在GPU上
# 层数越多，GPU利用率越高，推理越快

# 全部层放GPU（显存够的话）
llama-server -hf ggml-org/gemma-4-26B-A4B-it-GGUF:Q4_K_M -ngl 999

# 只放30层到GPU，剩下的跑CPU
llama-server -hf ggml-org/gemma-4-26B-A4B-it-GGUF:Q4_K_M -ngl 30

# 全部跑CPU（没有独显的时候）
llama-server -hf ggml-org/gemma-4-26B-A4B-it-GGUF:Q4_K_M -ngl 0
```

-ngl的值怎么定？从高往低试。先设个大数字尽量全上GPU，报CUDA out of memory就往下减，每次减10层，直到稳定。

大致参考：8GB显存（RTX 4060/3060）大概放20-25层Q4_K_M。12GB显存（RTX 4070）能放更多。24GB显存（RTX 4090/3090）基本全放进去。

性能监控

怎么判断模型跑得好不好？核心指标就一个：**tok/s（每秒生成的token数）**。

Ollama在每次对话结束时会打印。llama.cpp在日志里输出。LM Studio在界面底部实时显示。

tok/s	体感	适用场景
< 3	明显在等，有点煎熬	长文生成可以忍，交互式对话太慢
5 - 10	能用，但需要耐心	交互式对话的最低门槛
10 - 20	比较流畅	日常使用的舒适区
> 20	感觉不到延迟	实时对话、代码补全

M4 Pro 24GB上跑26B A4B Q4_K_M，实测大约15-20 tok/s。属于「比较流畅」这档。一段200字的中文回复等几秒，不快，但不至于失去耐心。

> **核心建议**
> 调优的核心原则：先跑起来，再慢慢调。别一上来就追求完美参数。用默认配置先感受效果，发现具体问题了再针对性调整。temperature调低一点、上下文开大一点、repeat_penalty提高一点。每次只改一个变量，才知道是哪个参数起了作用。

Gemma 4 完全指南

AI编程：从入门到精通

花叔 • AI进化论-花生

AppStore 付费 app 总榜第一「小猫补光灯」作者

《一本书玩转 DeepSeek》作者

加入知识星球 →

B站：AI进化论-花生 · 公众号：花叔 · X/Twitter · YouTube · 小红书 · 官网

Created by 花叔 • v1.0 • 2026年

本文档在 Claude Code 辅助下整理编写，内容仅供参考。