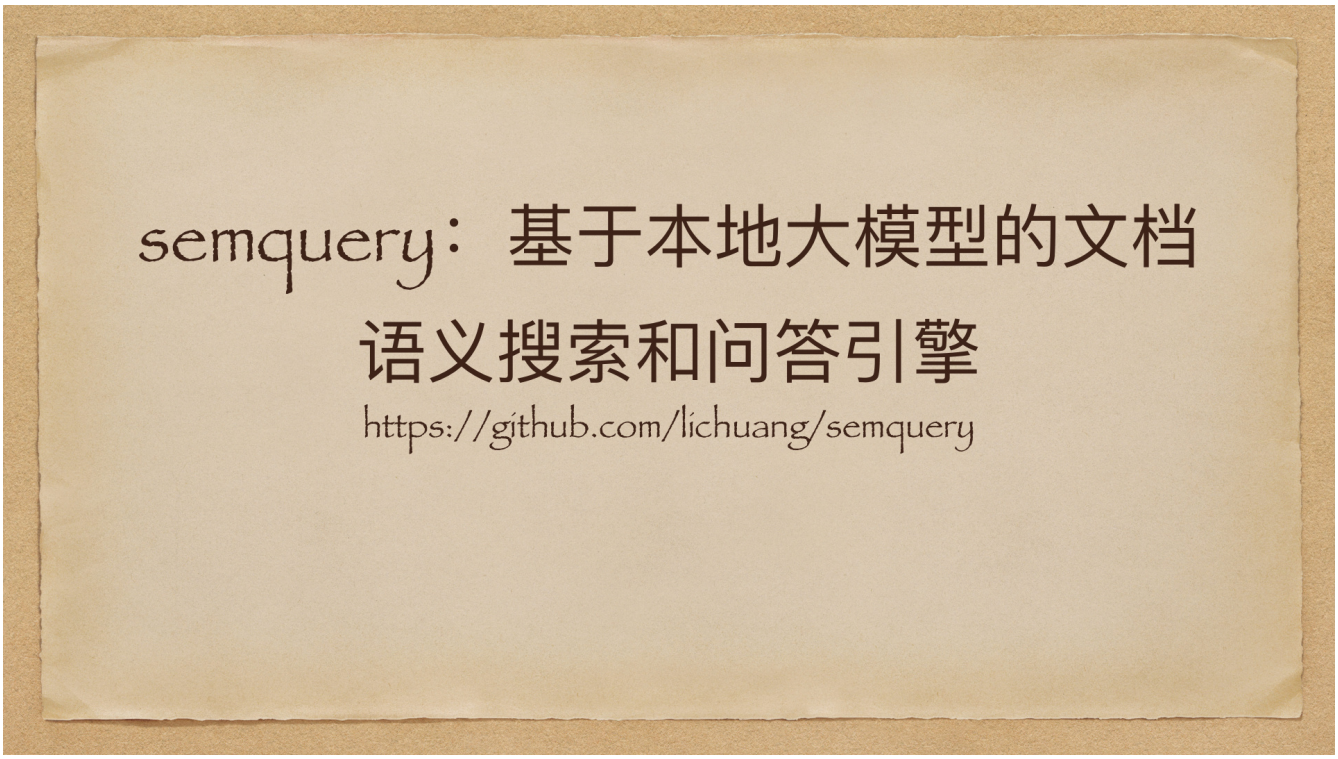
# semquery：基于本地大模型的文档语义搜索和问答引擎

codedump · 2026-09-29 · 演讲与问答全文

根据 2026 年 9 月 29 日的演讲与问答整理。保留第一人称表达及交流顺序，校订口头重复、断句与明显识别错误。性能数据、产品状态与判断以分享时点为准；配图来自讲者提供的幻灯片及现场演示。

## 项目名称与定位




项目名称（讲者提供版幻灯片，第 1 页）

00:00:00 · codedump

今天给大家讲一下我这个项目。它现在的名字叫 semantic query，原来叫 document query。当时我把项目简写为 DocQ，取的是 document 的前三个字母和 query 的第一个字母。

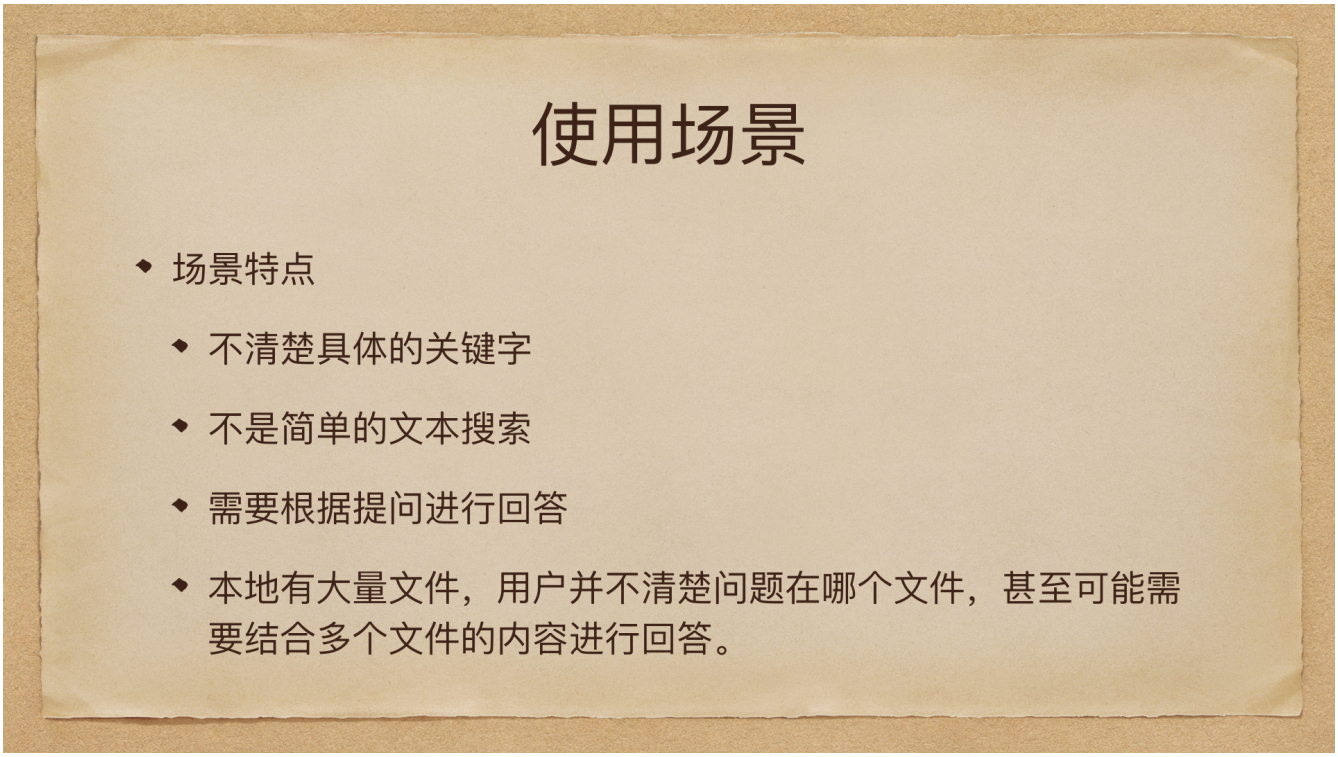
00:00:30 · codedump

后来发现有同名的 App。一个朋友建议我最好改一下名字，因为搜索引擎对一个新出现的同名项目可能不太友好，所以后来改成了 semantic query，semantic 就是“语义”的意思。其实这两个名字里，document query 更偏向产品，semantic query 更偏向一个技术名词。

00:01:08 · codedump

后来我想了一下，这个项目本身是底层的搜索和问答引擎。作为开源项目，它更多面向开发者，所以叫 semantic query，理解起来应该不会有太大问题。等我最后发布面向终端用户的产品时，肯定不会叫这个名字。

# 使用场景：在大量本地文档中提问




使用场景（讲者提供版幻灯片，第 2 页）

00:01:42 · codedump

但底层的问答引擎，我暂时还是定为 semantic query，也就是 semquery。首先讲一下使用场景。很多知识工作者，比如律师、医生，还有我太太，她是编剧，都有一个特点：私人设备上有大量私人文档，可能积累了十几万、二十几万的内容，也可能有成百上千份文件。

00:02:30 · codedump

比如他们要读的论文、我太太写的剧本、律师的法律文档、处理过的案件，以及中间记的一些笔记。即便不是知识工作者，平时写日记，积累几年、十几年，也会有很多内容。面对大量本地文档，当他们想查找或提问时，可能一时半会儿想不出具体该用什么关键词。

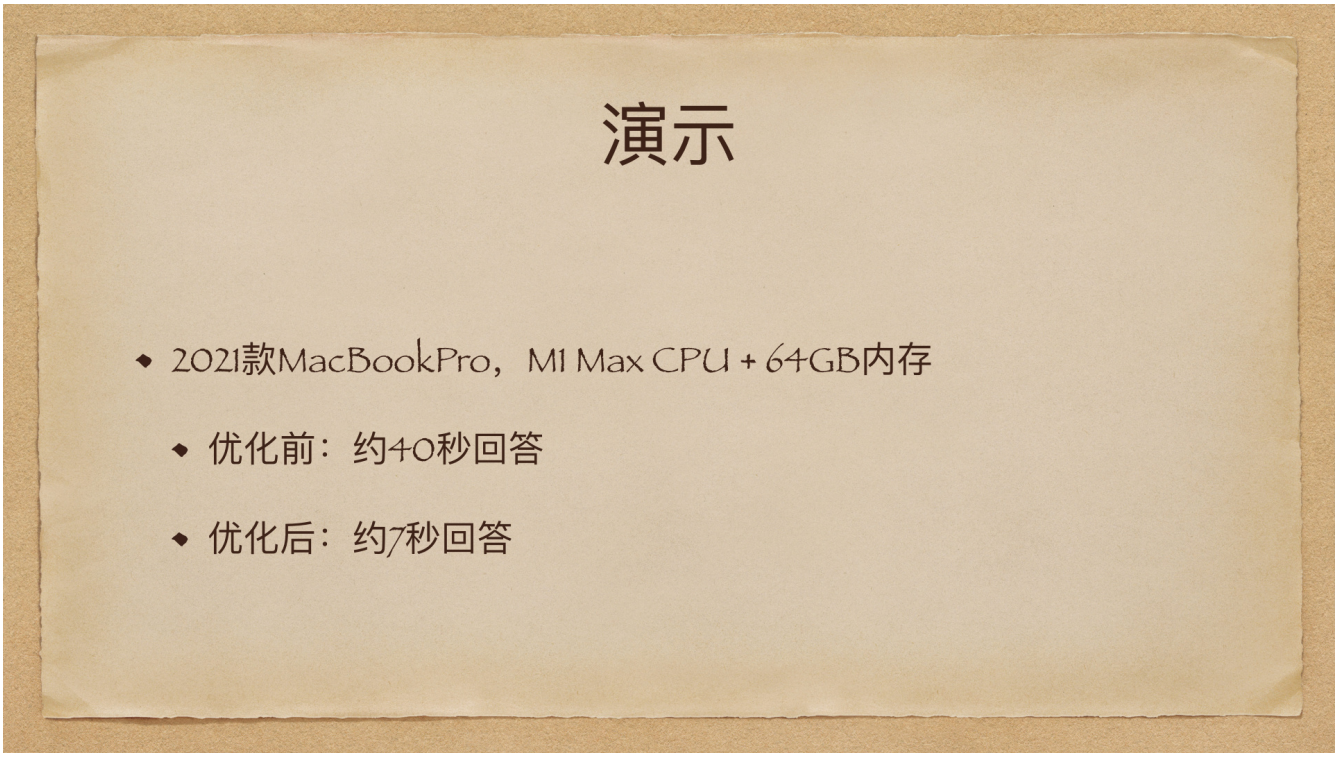
00:03:21 · codedump

第二，即便想到了关键词，用传统的文本搜索方法，也要求关键词能够精确匹配到那段文本，才有可能查到。第三，很多时候，除了搜索，还需要根据文档回答问题。

00:03:54 · codedump

比如律师可能会问：“我之前有没有处理过类似性质的案件？当时我是怎么做的？”这是一个问答流程，不是搜到文档就结束了。用户还希望对文档提问。这就是我当时想到的一个大的使用场景。

## 现场演示：初始化、索引、搜索与回答




演示（讲者提供版幻灯片，第 3 页）

00:04:24 · codedump

在继续往下讲之前，我先演示一下目前的项目。有了这个演示，大家理解后面的讲解可能会更直观一点。我先把屏幕切回去。我分的这两个屏幕都可以看到，没问题吧？

00:04:52 · 观众

可以的。

00:04:55 · codedump

这是项目目录，配置和日志文件目前都存在这个目录下面。我先把里面的文件都删掉，然后对项目做一次初始化。初始化时，它会创建数据库里的数据表。比如下面这个 semq.db，就是存放项目数据的 SQLite 数据库。目前我只用了 SQLite。因为还会涉及文本搜索和向量搜索，我当时考虑过要不要再引入一个专门做嵌入式向量搜索的数据库。

00:06:06 · codedump

比如 LanceDB。但我后来发现一个痛点：数据库里其实存着几种类型的数据。第一种是项目本身的元数据，比如每个文档对应的文档 ID，以及它们之间的关系。其次是向量数据，还有用于 FTS 文本搜索的数据。

00:06:44 · codedump

比如现在新增一个文档，就要同时增加这三种数据。如果再引入 SQLite 之外的数据库，比如 LanceDB，就没有办法直接保证同时更新这三种数据时的事务性。我同时写两种不同的数据库，又要保证事务性，可能还得做很多工作。

00:07:24 · codedump

所以一开始选型时，我就从简单出发，先选用了 SQLite。SQLite 也有相应的扩展，可以支持文本搜索和向量搜索。只不过它的向量搜索可能比较简单、直接，是 flat 级别的那种搜索。这就是第一步，先初始化项目。

00:08:06 · codedump

为了测试，我在项目里加了一个 testdata 目录，里面有一些 Markdown 文件。这些文件取自右边分屏展示的、我博客上的分布式系统教程。教程正文一共有七章。我先用 LaTeX 写，写完之后再转成 Markdown。

00:08:41 · codedump

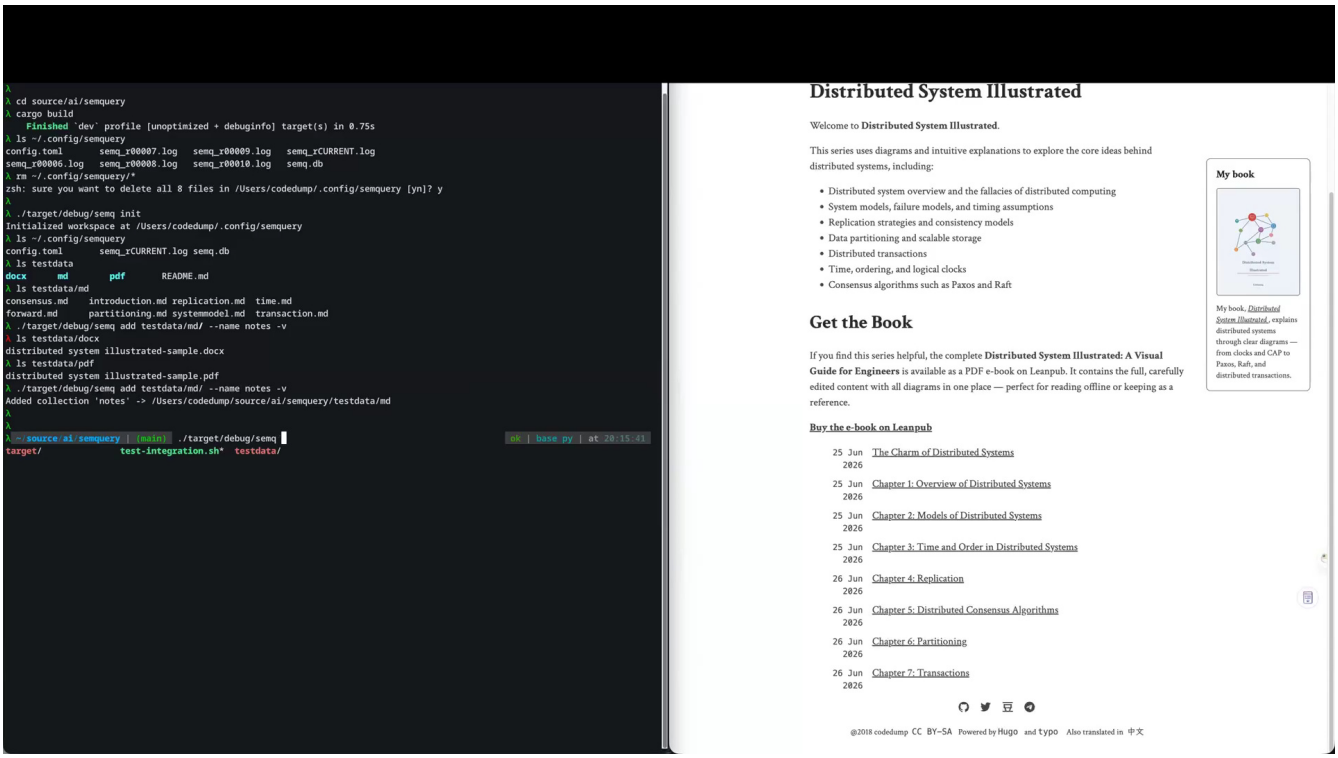
我的博客是用 Hugo 建的，所以可以拿这些 Markdown 渲染成网页。除了前言，还有七章正文，一共八个章节。我把它们的 Markdown 文件都放进了 testdata 目录。当时大概算了一下，这些文件一共约有 18 万个英文字符。

00:09:17 · codedump

现在就可以把整个目录加进项目。你们也能看到，testdata 里还有 docx 和 pdf，这两个目录用来验证项目对 Word 和 PDF 的支持。不过这两种格式里放的都是文档的 sample 部分，没有完整的章节。

00:09:57 · codedump

所以测试时，我还是使用 Markdown 部分。命令行支持一个 -v 参数，可以把整个流程显示得更详细。我先把目录加进去，然后就可以对它进行索引。




现场演示：初始化与加入文档目录（成片 10:20 截图）

00:10:21 · codedump

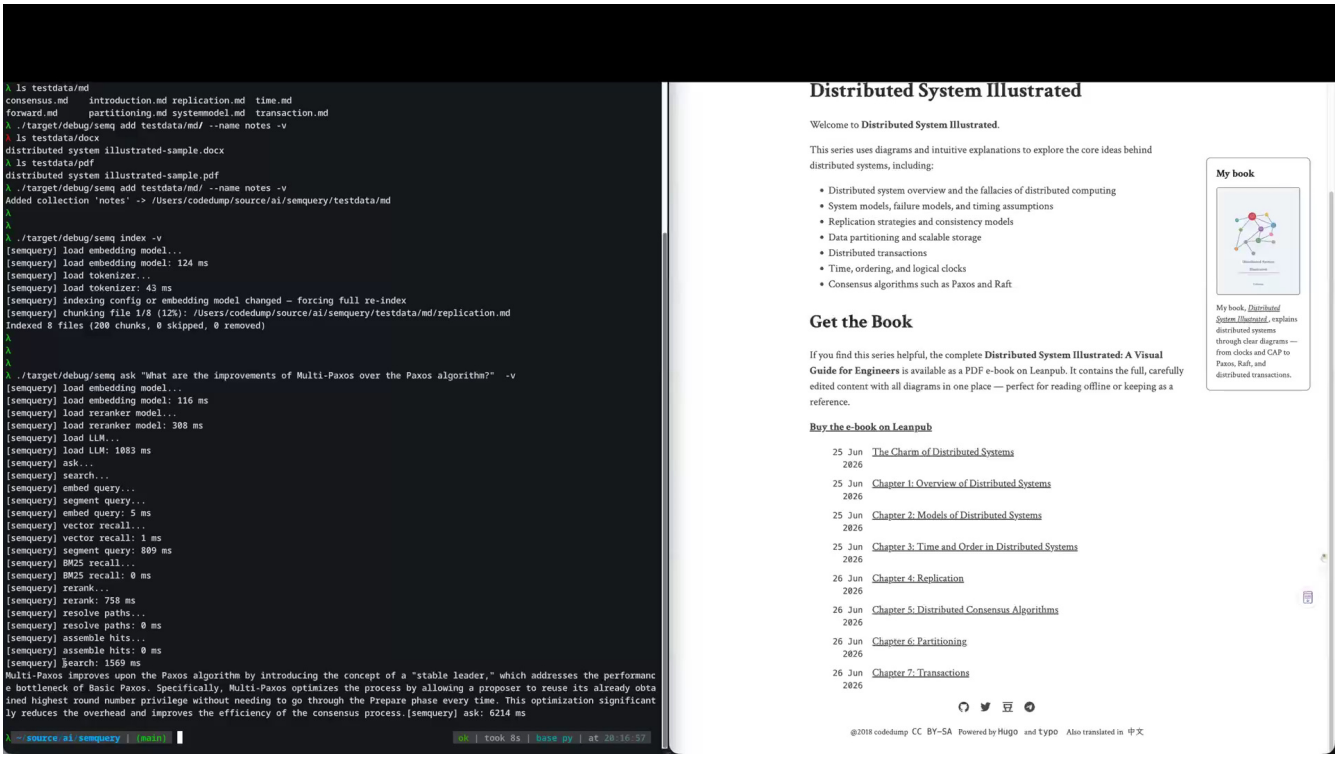
建立索引时也加上 -v。这个流程会对刚才加入的 testdata/md 目录下的所有文件建立索引，并把文件分成多个 chunk。后面讲技术实现时，会专门讲这一部分。这里显示已经索引了八个文件，同时产生了 200 个 chunk。

00:10:56 · codedump

有了这些索引，就可以对文档提问。比如分布式系统文档里提到了 Paxos 算法，我就问它：“Multi-Paxos 算法相比 Paxos 算法有哪些改进？”加上 -v 后，可以显示每个流程花了多少时间。

00:11:32 · codedump

这里首先加载 embedding 模型，然后加载 rerank 模型，分别花了这些时间。这里没有换行，我指一下最花时间的部分：先在索引里查询这些分块，再把查到的分块放进 rerank 模型做重排序。




现场演示：索引、模型加载与查询耗时（成片 11:45 截图）

00:12:04 · codedump

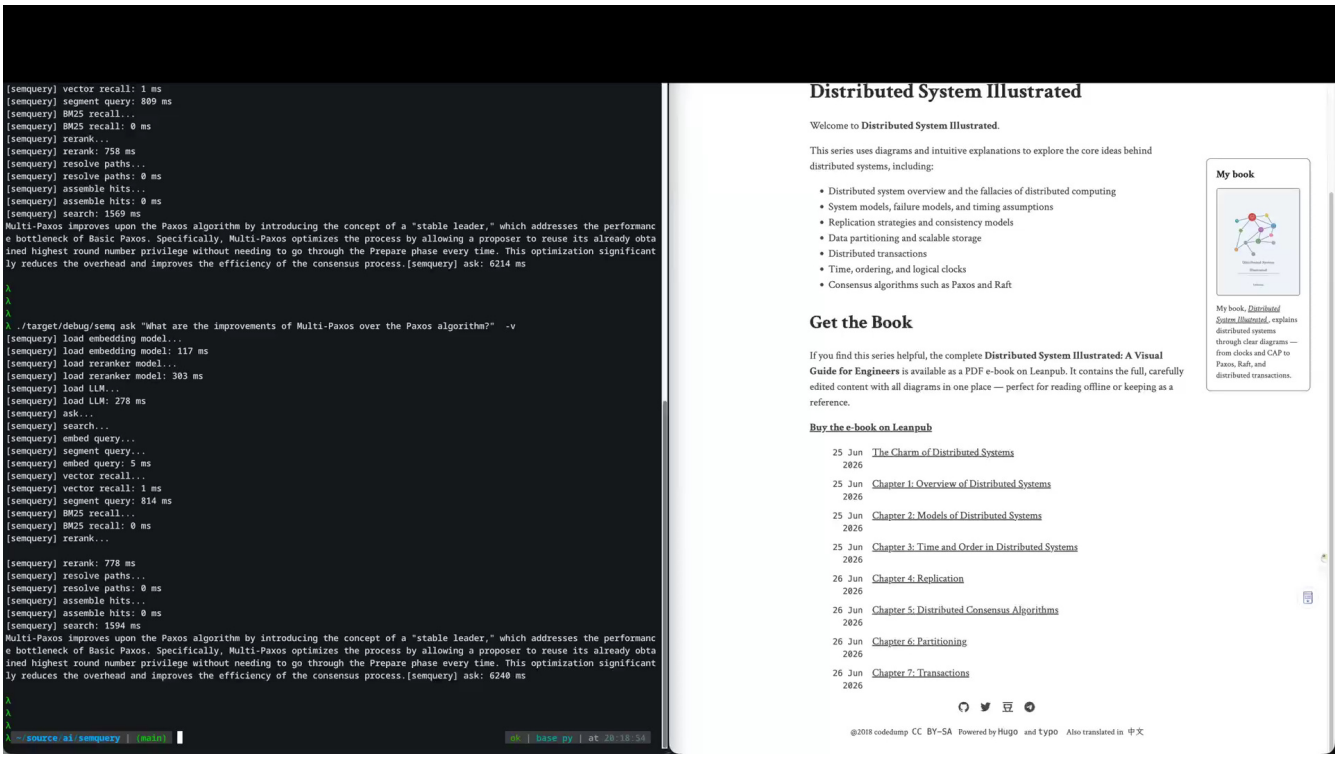
到查询完成，大概花了 1,500 多毫秒，也就是一秒多。最后把查询结果放进本地 LLM，生成问答结果，这一步花了大概 6,000 多毫秒。整体下来，整个流程花了七到八秒。

00:12:37 · codedump

不知道大家有没有注意到，刚才吐出问答结果时，它是一个流式的过程。不是等到完整答案出来后，再一次性显示。这也是当时做的一个优化，等会儿可以聊。我们可以再看一下，注意这个流式输出。

00:13:07 · codedump

最开始做的时候，时间还蛮长，大概四五十秒才能出结果。所以我做了一个体验上的优化，把它改成流式输出，不必等整个结果完成才显示。刚才演示的是提问，而提问背后依赖的是搜索结果。




现场演示：回答与流式输出（成片 13:30 截图）

00:13:39 · codedump

我们也可以直接执行 search。搜索结果有点多，所以我加了 more。它会把文档里与问题最相关的几个部分打印出来。这是引用的第一部分，还有第二部分。前面问答的效果，依赖于这里的搜索结果：先找到内容，再把它喂给 LLM，最后生成回答。

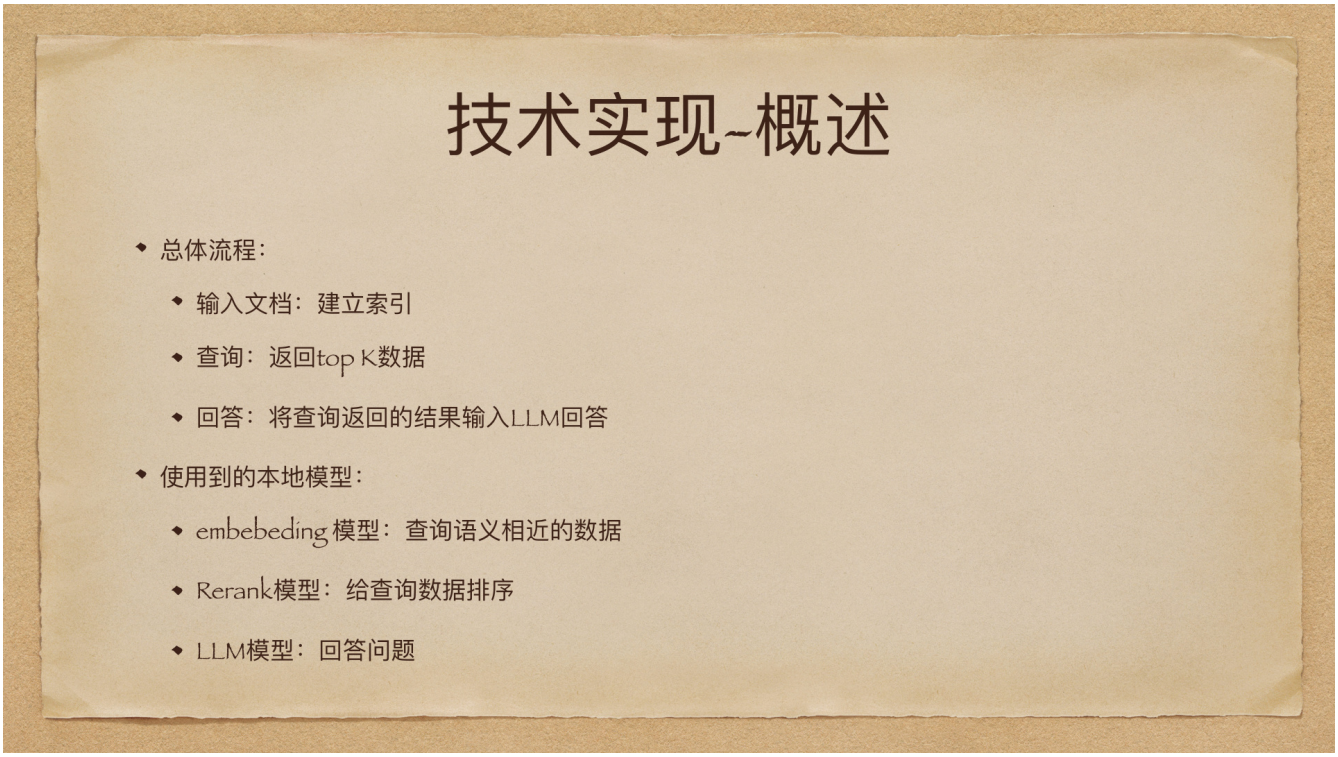
00:14:19 · codedump

刚才演示了项目怎么使用。在我自己的 MacBook Pro 上，问答大概花七到八秒。我的机器是几年前的 MacBook，用的是苹果 M1 Max。

00:14:51 · codedump

内存是 64 GB。因为我写 Rust，编译比较慢，当时就尽量选了内存大一点的机器。现在是刚才这个效果；优化之前大概需要四五十秒，我觉得可用性就不是很够。优化之后，从时间和效果上看，我觉得还可以。接下来讲一下目前的技术实现。

# 技术实现概述与三个本地模型




技术实现概述（讲者提供版幻灯片，第 4 页）

00:15:25 · codedump

整体分为三个流程：输入文档、查询和回答。输入文档后，先建立索引；查询依赖索引，返回与问题最匹配的 top K 数据；再把这些数据和问题一起输入本地 LLM，得到回答。

00:16:06 · codedump

在这三个步骤里，查询这一步是最重的。查询完数据后，再输入 LLM，相对来说反而轻量一点。

00:16:19 · codedump

用到的本地模型有三种。第一种是 embedding 模型，负责把输入文本转成向量，辅助查询语义相近的数据。第二种是 rerank 模型，负责对查询结果排序。因为不管用文本搜索还是向量搜索，查出来后还要筛选，比如先查到 100 条。

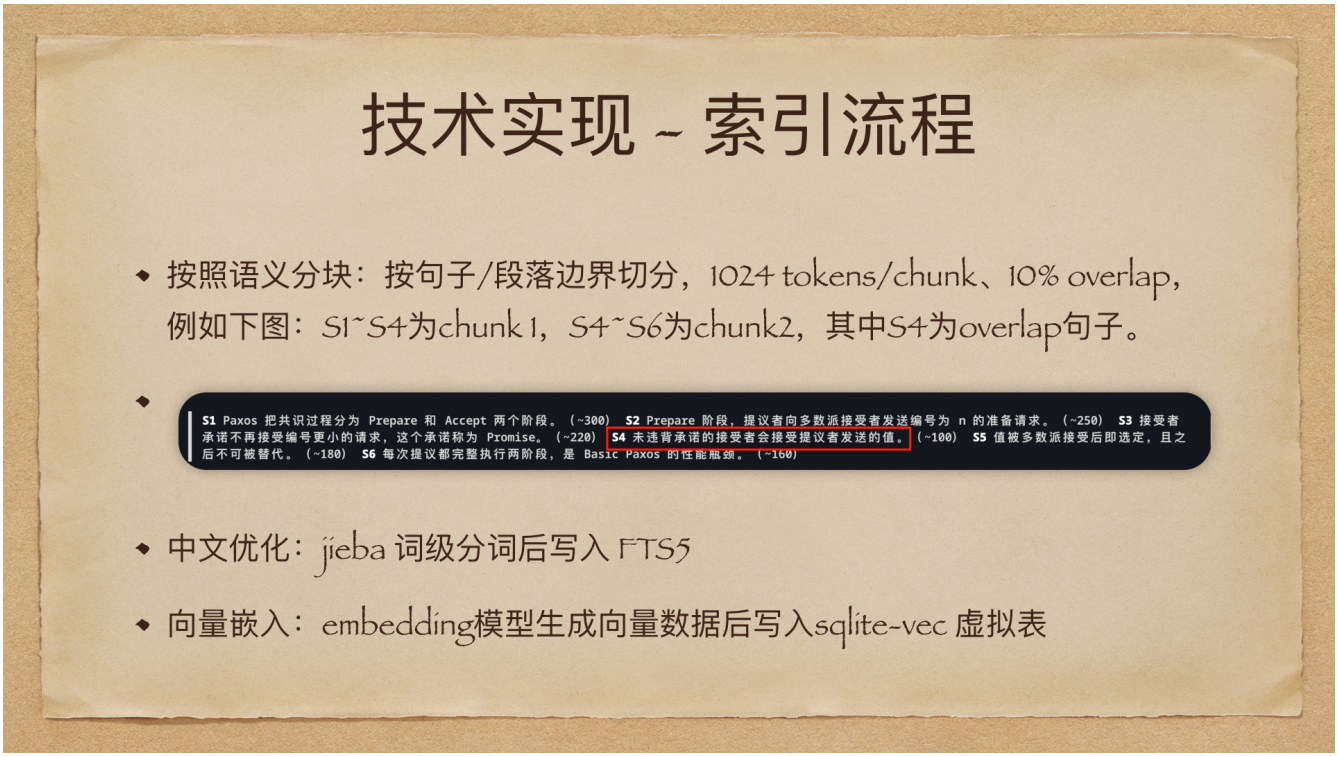
00:17:02 · codedump

再从中取分数最高的 20 条，输入 rerank 模型排序。等会儿会仔细讲这一部分。最后是本地 LLM，目前用的是千问的一个模型。这三个模型不算特别大，千问那个稍大一点，大约 2.7 GB，整体还好。

00:17:32 · codedump

第一次使用时，项目会先检查本地有没有这几个模型。有的话就继续沿用，没有的话才下载，不会每次使用都重新下载。这就是整体流程和用到的本地模型。

# 索引：语义分块、中文分词与向量存储




索引流程：语义分块（讲者提供版幻灯片，第 5 页）

00:18:02 · codedump

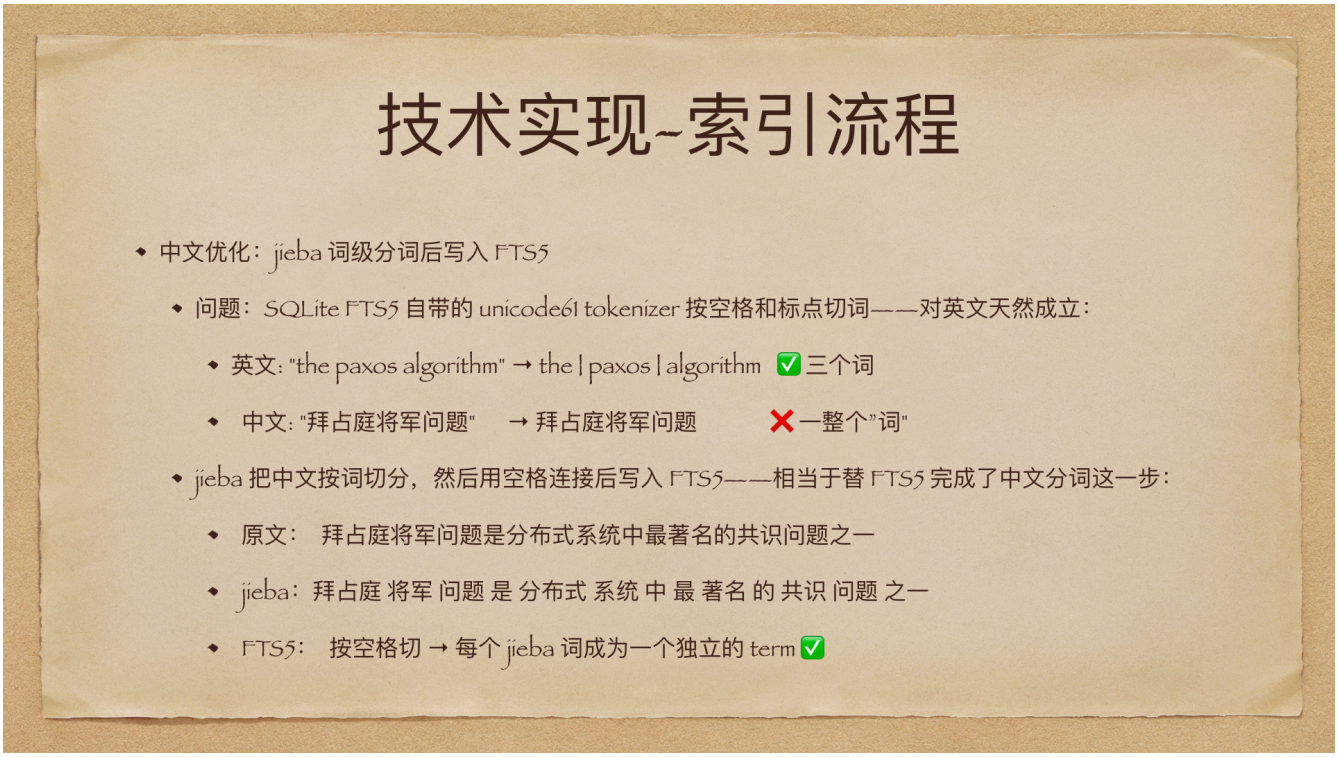
下面按这三个流程分别讲：先讲索引，再讲查询，最后讲问答。索引首先要按照句子、语义边界进行切分，不能切在句子中间。比如一句话有十个字符，不能在第五、第六个字符之间就切成一个 chunk，否则句子就不完整了。

00:18:43 · codedump

第二个问题是，不同 chunk 之间还要有 overlap，这样衔接会好一点。看下面这个例子，文本有六个句子，我标成了 S1、S2、S3、S4、S5、S6。

00:19:14 · codedump

切分时以句子为边界，但不是一个 chunk 对应一个句子，一个 chunk 里可能有多个句子。目前我设定一个 chunk 大约是 1,024 个 token。以这六个句子为例，把第一句到第四句分为 chunk 1，第四句到第六句分为 chunk 2。




索引流程：中文分词（讲者提供版幻灯片，第 6 页）

00:19:47 · codedump

第四句，也就是红框标出的那句，是 overlap 的部分，让两个 chunk 衔接得更好。索引里的第二个问题是中文优化。SQLite 自带的文本搜索，按空格和标点来切词。比如这里的 “the Paxos algorithm”。

00:20:21 · codedump

英文单词之间有空格、有标点，对这个搜索引擎很友好，可以切成三个词。但中文的“拜占庭将军问题”，这些词连在一起，没有空格或标点作为明确的区分。

00:20:54 · codedump

所以在把文本写入 SQLite 的全文搜索之前，我先用 jieba 做中文分词，再把分词结果写入 FTS。相当于在 SQLite FTS 原本的分词流程前，补上中文分词这一步。

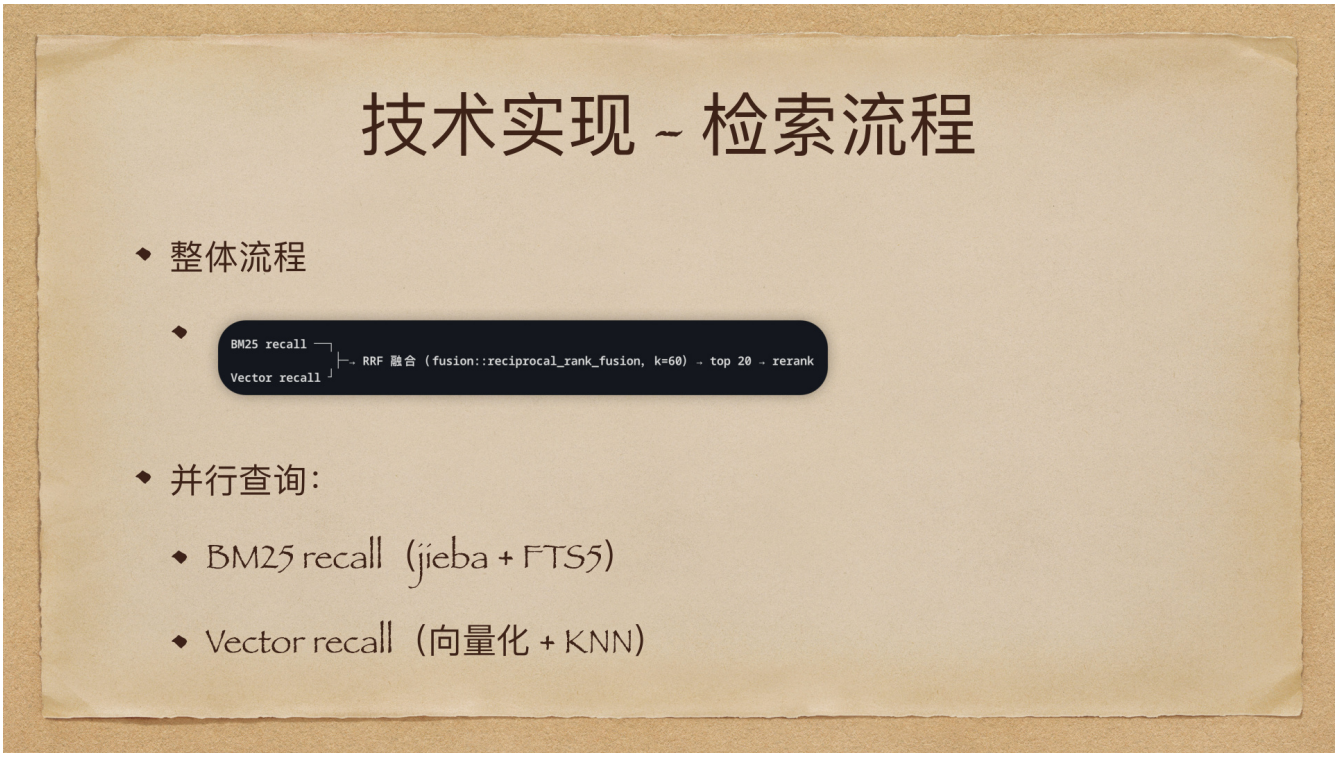
00:21:27 · codedump

先帮它把词切出来，再输入 FTS。这是索引流程里为了面向中文做的优化。最后是向量嵌入：把文本输入 embedding 模型，生成向量数据。

00:21:58 · codedump

向量数据也写入 SQLite，使用的是专门做向量存储的 sqlite-vec 扩展。索引流程大概就是语义分块、中文优化，以及根据输入文本生成并存储向量。接下来是检索，它是这三个流程里最复杂的一个。

# 混合检索与 RRF 融合



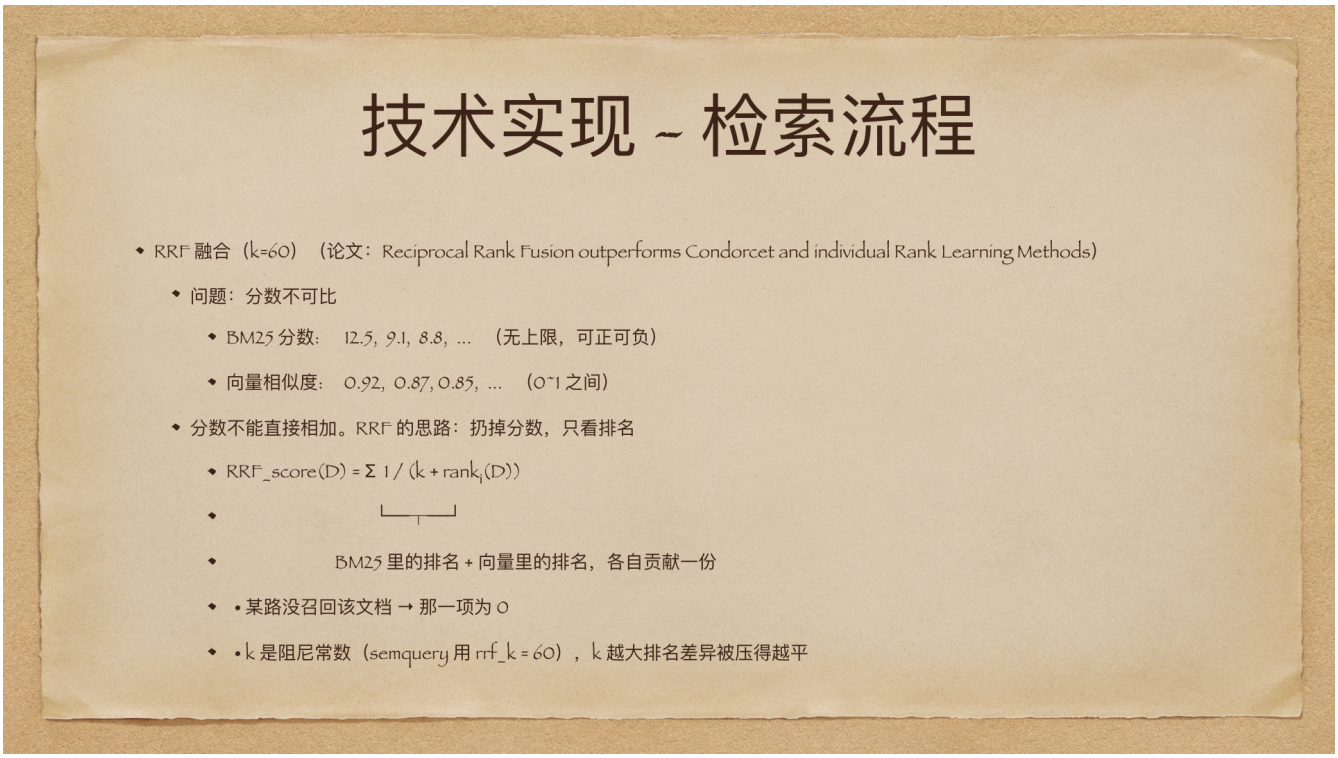

检索流程：并行召回（讲者提供版幻灯片，第 7 页）

00:22:31 · codedump

检索整体上分为三步。第一步是 recall，也就是召回，包括向量召回和 BM25 召回。BM25 这一路使用 jieba 和 FTS，根据问题到前面存储的全文索引里做文本搜索。

00:23:08 · codedump

另一条路是 vector recall，也就是向量搜索。这两路完成后，再把结果交给 RRF 做融合。RRF 来自这里这篇论文，它要解决的问题是：BM25 和向量相似度的分数并不可直接比较。




检索流程：RRF（讲者提供版幻灯片，第 8 页）

00:23:39 · codedump

BM25 的分数没有固定上限，可以是正的，也可以是负的。这里的向量相似度则用 0 到 1 的浮点数表示，有点像百分比的概念。所以拿到这两路查询结果和分数后，不能直接把分数相加。

00:24:13 · codedump

RRF 这篇论文提出的方法是：不直接加分数，而是看排名。我举个例子，比如河南高考和海南高考的分数就不一样。我是海南人，不能简单拿两个地方学生的原始分数，判断谁比谁怎么样。

00:24:47 · codedump

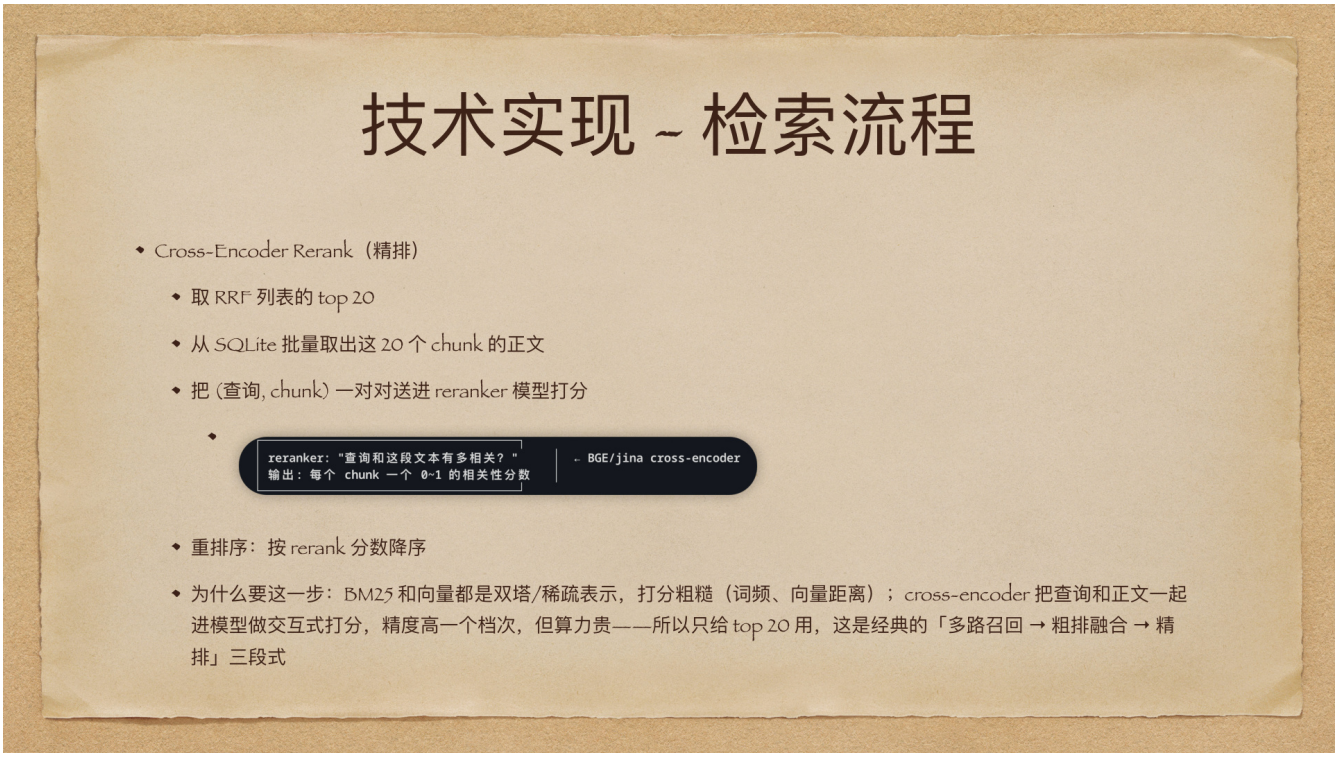
它的思路就是不看原始分数，只看排名。论文里给出了计算 RRF 分数的公式，把多路搜索的排名放在一起，合成一个分数。这里还有一个常数 k。

00:25:20 · codedump

目前我按照论文取 k 为 60，它叫阻尼常数。这个参数我其实也不太说得明白，大概理解是把排名上的差异压平；如果排名差得不大，压平的程度也不一样。有兴趣可以再看一下这个公式和论文。

> 编者注：此处保留讲者对参数尚不完全确定的表述。幻灯片中的 RRF 公式为 Σ 1 / (k + rank_i(D))，其中 k = 60；讲者提供的说明为“k 越大，排名差异被压得越平”。

# Cross-Encoder 重排序与问答流程




检索流程：Cross-Encoder（讲者提供版幻灯片，第 9 页）

00:25:54 · codedump

经过 RRF 融合，已经得到排名靠前的一些 chunk，但还需要再解决一个问题：前面的文本搜索和向量搜索，对查询与正文之间的交互处理还比较粗糙。

00:26:35 · codedump

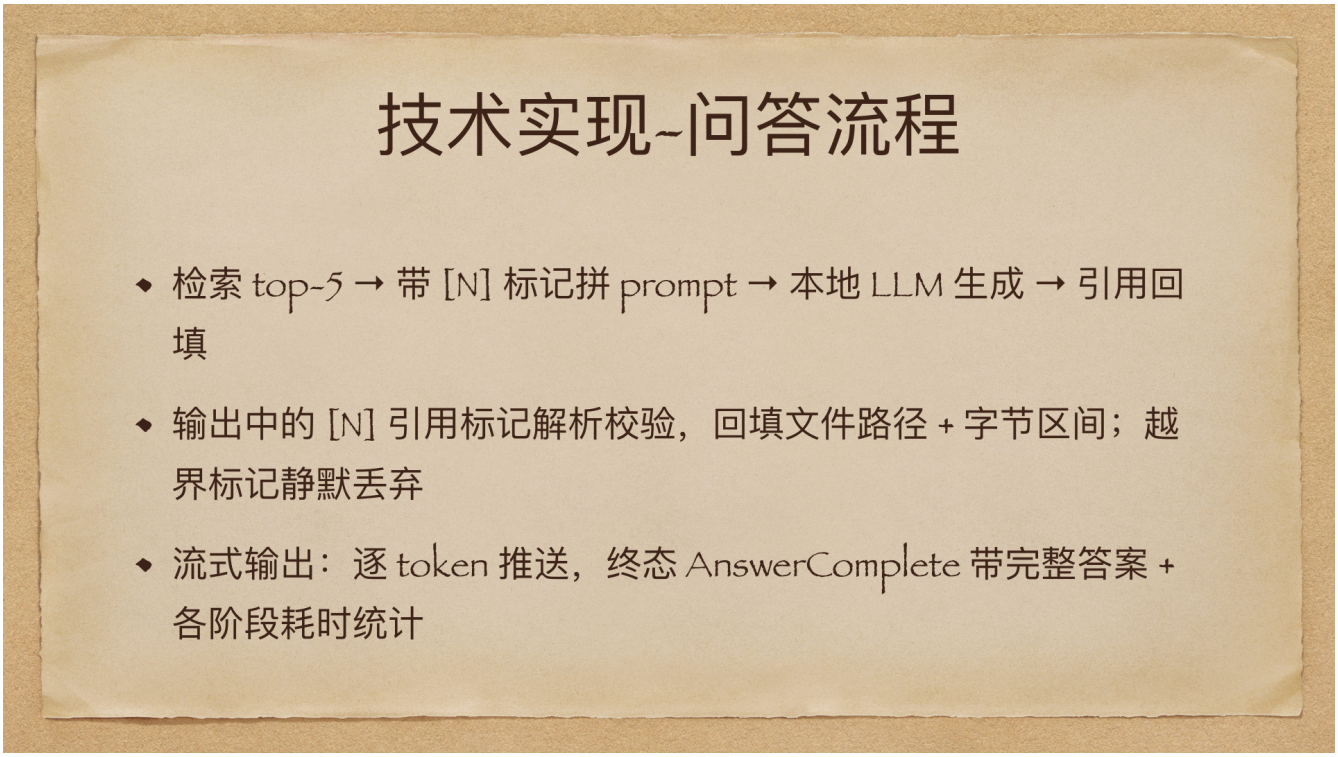
所以最后还要做 rerank，把提问和 chunk 组成一个二元组，一起输入 rerank 模型打分。举个例子，BM25 和向量搜索各取前 100 条，先找出各自认为与查询相关的数据。

00:27:14 · codedump

RRF 再从这两路召回的结果中取前 20 条。我还要把这 20 条放进 rerank 模型，再打一次分。BM25 主要做字面统计，没有这种语义交互；向量召回同样也有它自身的问题。

00:27:47 · codedump

我看到有些其他产品还会做图数据搜索。在多路召回里，不同路返回的分值可能不一样。最后还是要结合查询，用 rerank 模型对返回的文本再做一次打分。




问答流程（讲者提供版幻灯片，第 10 页）

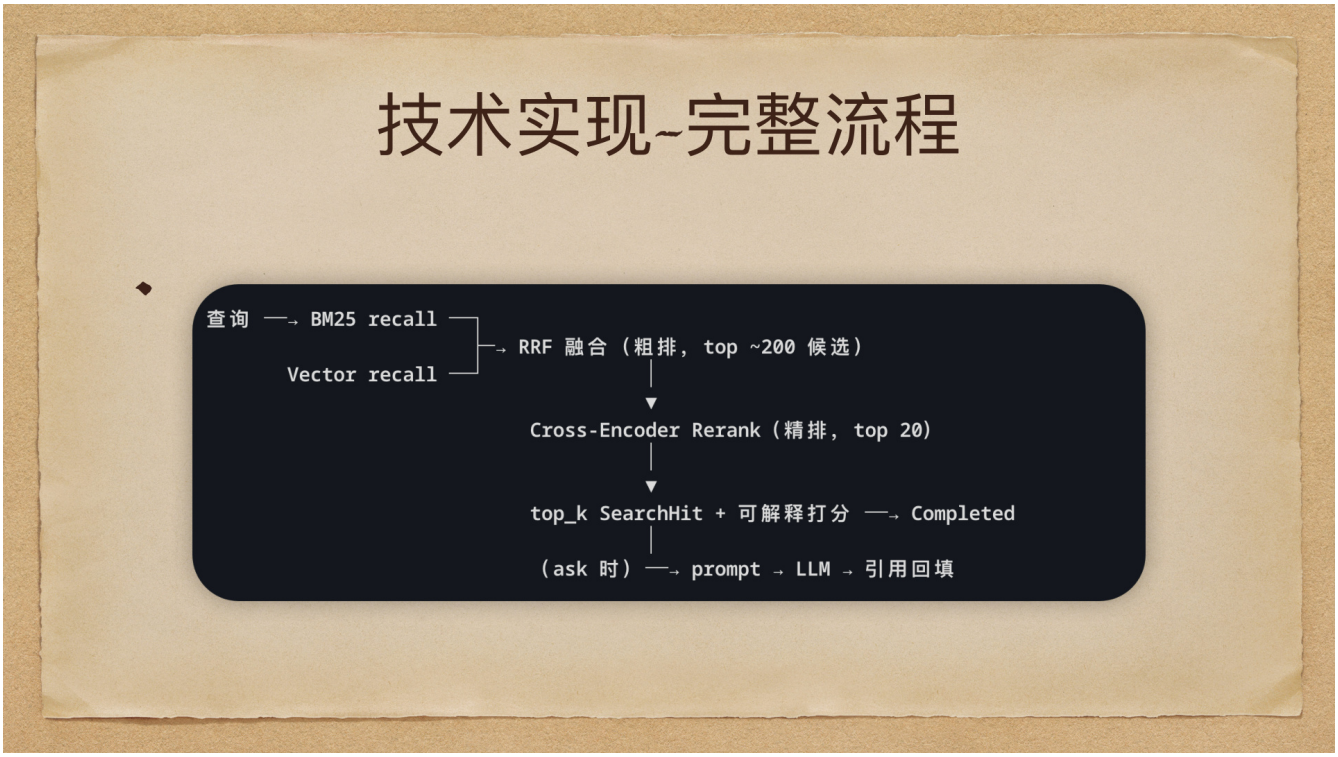
00:28:19 · codedump

输入 rerank 的，是经过 RRF 筛出的前 20 条数据。rerank 再打分后，我目前取分数最高的五个 chunk。把这 top 5 的内容拼成提示词，输入本地 LLM，就可以生成回答。

00:28:56 · codedump

这就是整体流程。查询时，先做 BM25 和向量召回，再把两路结果交给 RRF 融合，做一次粗排。RRF 的输入可能有前 200 条数据，融合后只取前 20 条交给 rerank 模型。

## 完整流程与性能优化




完整流程（讲者提供版幻灯片，第 11 页）

00:29:37 · codedump

再经过 rerank 排序，取前五条，拼成提示词输入本地 LLM，生成回答。这就是完整流程。不过在最开始做技术选型和实现时，整体耗时并不太理想。




性能优化（讲者提供版幻灯片，第 12 页）

00:30:22 · codedump

一开始整个问答流程需要四五十秒，我做了几个优化。第一，BM25 召回和 vector 召回可以并行，最开始我把它们写成了串行。既然最后只是把两路结果合起来输入 RRF，就可以并行处理。所以第一个优化是把多路召回并行化。

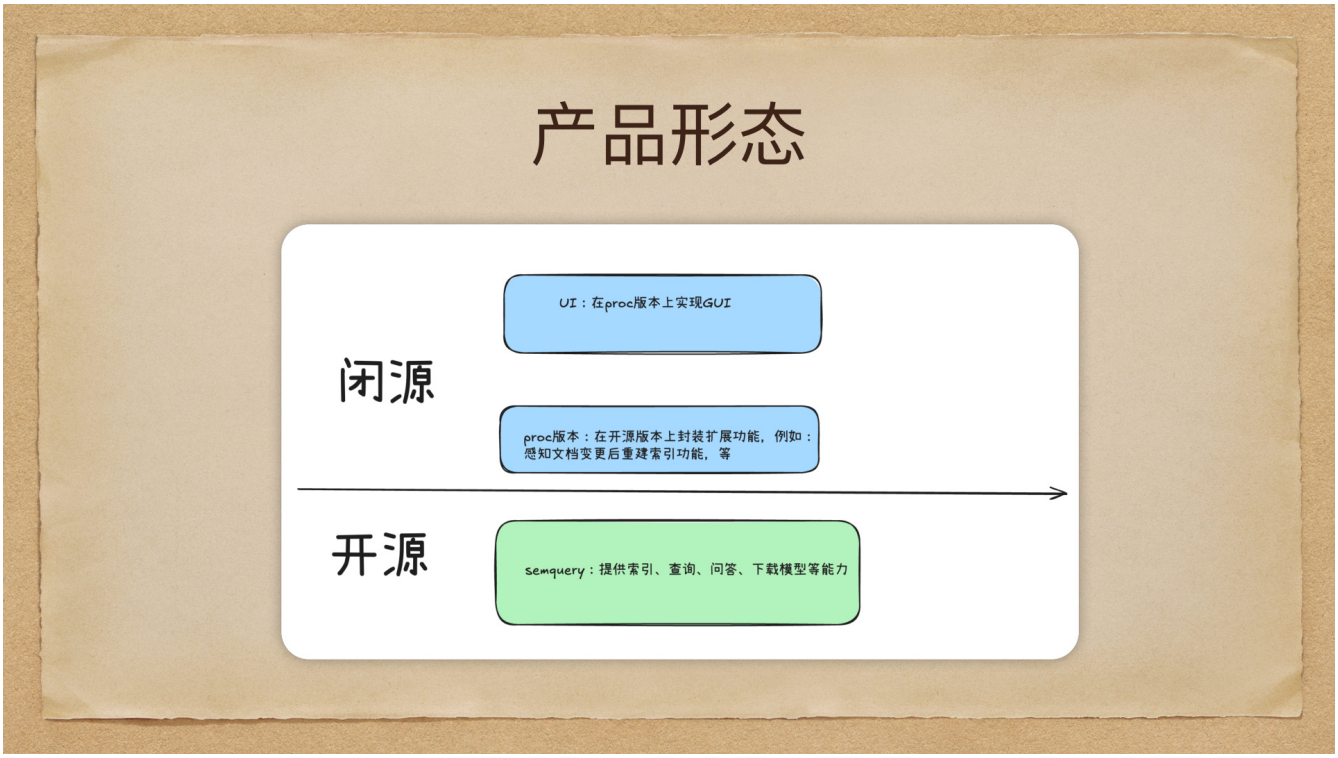
00:30:56 · codedump

第二个优化与苹果的 GPU 有关。我用的 llama.cpp 在当时的编译配置里，没有默认启用苹果的 GPU，所以编译时要加上相应选项。可以看一下，把 Metal 这个开关打开后，速度一下子提升了不少。这是把几十秒优化到十秒以内贡献最大的一步。第三，如果需要等几十秒才给用户反馈，体验会很不好。

00:32:15 · codedump

所以后来又做了流式输出的优化。需要与模型交互的这些流程，尽量变成逐 token 输出：即便整个过程还慢，只要吐出一个 token，就马上显示到屏幕上，用户的感知会好一点。这几个优化做完后，就从几十秒到了几秒钟。

## 产品形态：开源引擎与闭源扩展




产品形态（讲者提供版幻灯片，第 13 页）

00:32:45 · codedump

我觉得可用性还可以。但这只是我机器上的效果，不同机器可能有不同表现，也是后面要改进的地方。再讲一下产品形态。目前我把它分为开源和闭源两部分。开源部分就是 semquery，提供索引、查询、回答和自动下载模型这些能力。

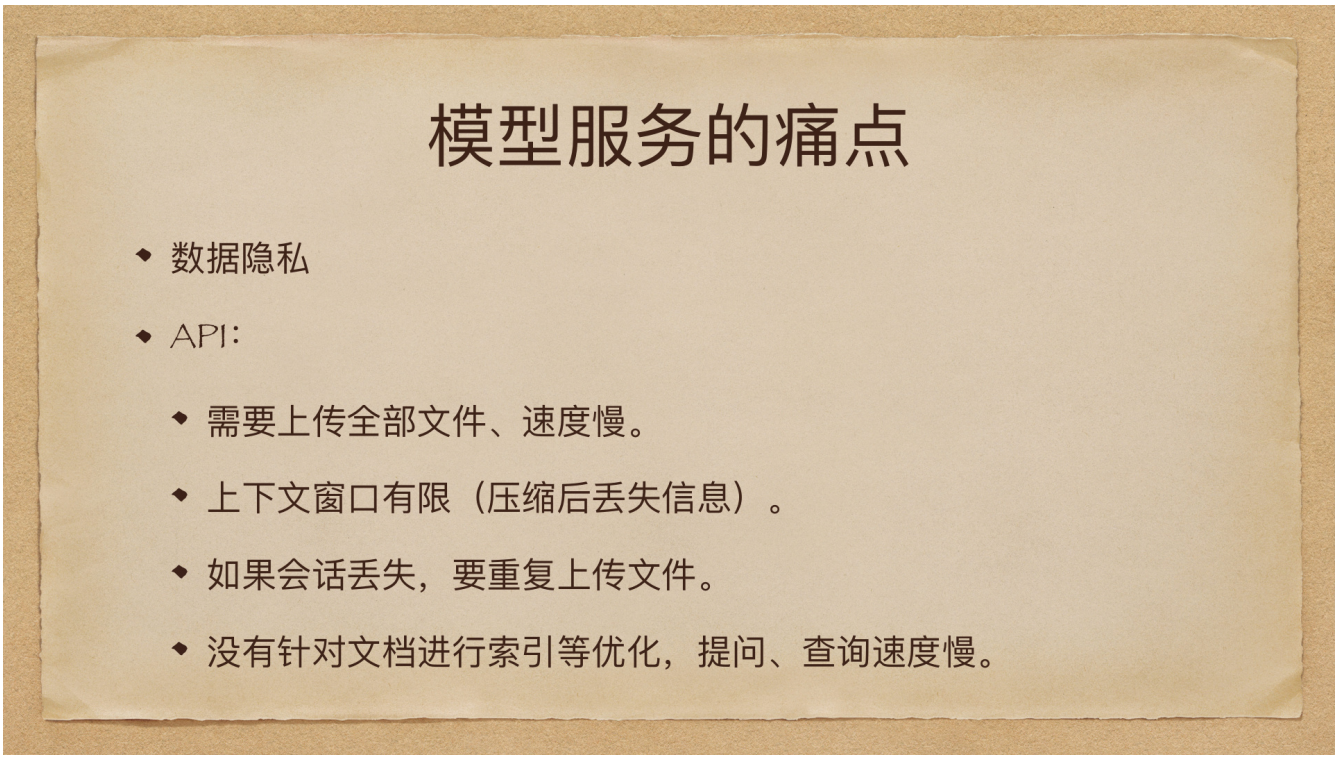
00:33:20 · codedump

这些能力是完全开放的。但刚才的演示也能看出，使用步骤还是有点麻烦。比如加入一个目录后，某个文件发生变化，就应该删除相关索引并重建。目前这个功能没有放在开源版本里，我打算放到闭源版本。

00:33:52 · codedump

闭源部分会有一个 Pro 版本，在开源能力上封装扩展功能，再基于 Pro 版本实现 GUI。整体大致分为开源和闭源两部分：最底层完全依赖开源部分，这一部分不会有保留；在它之上的一些扩展功能，包括 GUI，我会放在闭源部分。

## 本地模型、隐私与 API 服务




模型服务的痛点（讲者提供版幻灯片，第 14 页）

00:34:30 · codedump

这是目前设想的产品形态。接下来讲讲，为什么我觉得这个产品使用本地大模型会更好。首先当然是数据隐私。无论是医生、律师的资料，还是个人日记，如果要通过 LLM API 上传出去，大家都会想一想。

00:35:08 · codedump

很多企业也有本地部署大模型的需求。像去年春节 DeepSeek 爆火之后，就出现了一个生意：在企业内部，专门给它装一台能跑 DeepSeek 的机器。企业的要求就是数据不能出去，在本地运行即可。我觉得数据隐私在任何时候强调都不过分。

00:35:44 · codedump

第二，API 服务也有自身的问题。即便先不谈隐私，按照刚才的使用场景，用户本地有成百上千份文档，如果每次提问都上传全部文件，速度会很慢。当然，现在一些聊天工具，比如 WorkBuddy、Cherry Studio，可以把会话缓存下来。

00:36:22 · codedump

缓存会话 session 之后，可能只上传一次文件就够了。但 LLM API 的上下文窗口始终有限。内容太多时，你不知道它什么时候会对文档做压缩。会话如果丢失，还得重新上传。

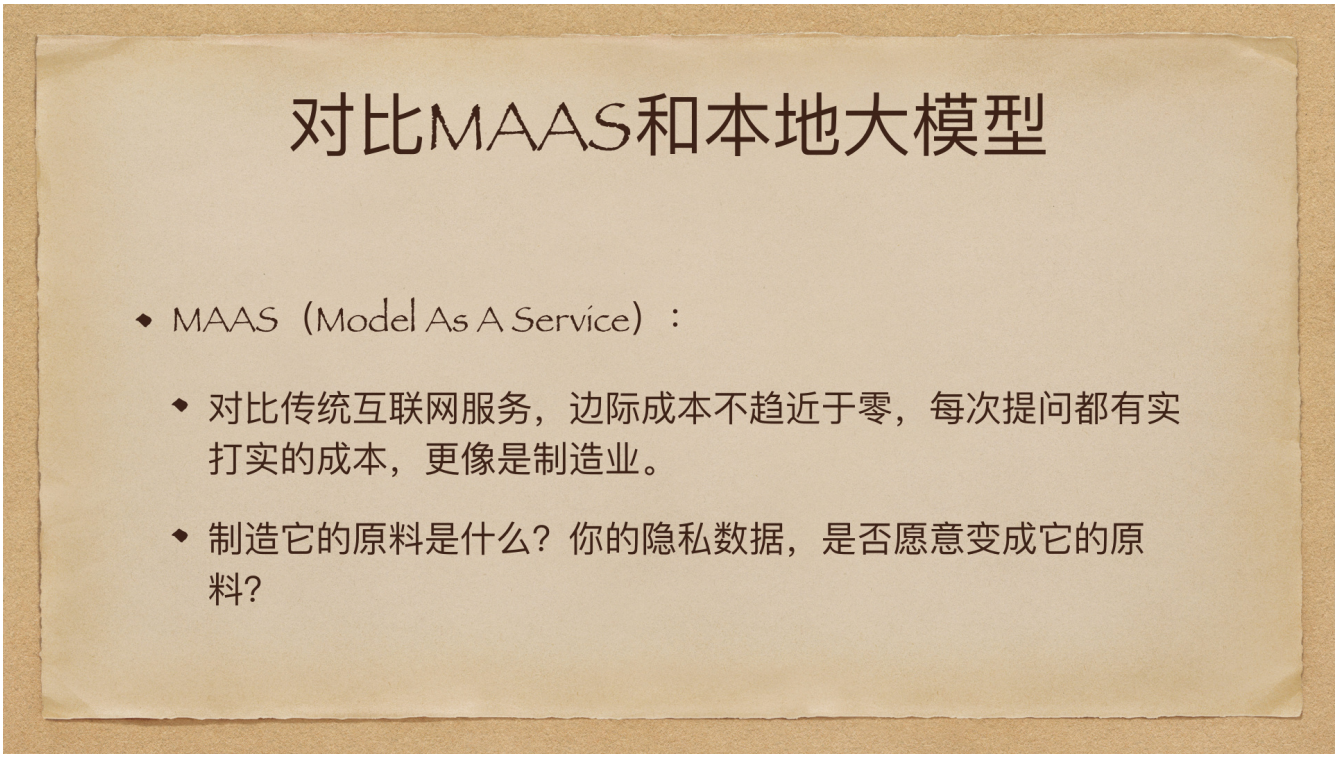
00:36:53 · codedump

另外，我刚才讲到索引时，其实已经做了很多事情，包括文本搜索、中文分词和向量存储，针对文档做了很多索引优化。直接把文档上传给 API，它并不会自动替你做这里这些索引优化。

00:37:25 · codedump

所以我的意思是，即便不考虑隐私，在这么多文档里，上传给 API 再提问、查询，速度和体验也可能不好。接下来对比一下 Model as a Service，也就是 MaaS。以前互联网里经常讲“某某 as a Service”，这是一种常见的产品形态。

## MaaS 的成本与本地模型的产品机会




MaaS 与本地模型：成本（讲者提供版幻灯片，第 15 页）

00:38:07 · codedump

打个比方，假设你想建自己的博客，前期可能要买域名，再把博客托管到云平台上，有一笔服务器成本。这是一开始的投入。博客建起来后，一个人、100 个人、1,000 个人，乃至 1 万个人来访问，对这个博客而言，成本并没有随每次访问明显增加。

00:38:54 · codedump

也就是说，以前互联网服务的一个特点，是服务的边际成本趋近于零。但到了 Model as a Service，大家用大模型都知道，每次都在消耗 token，各家的质量和 token 费用也不同。所以我觉得，MaaS 相比传统互联网服务，现在更像制造业。

00:39:31 · codedump

制造业是什么意思？比如做一个小玩具，或者早餐吃的一份糕点，哪怕供应链已经把成本压得很低，每多制造一份产品给用户，仍然有实打实的成本。它不像传统互联网服务那样，边际成本趋近于零。

00:40:03 · codedump

所以我感觉现在的 MaaS 更像制造业。这又回到了数据隐私的问题：如果它是制造业，制造它的原料是什么？像前阵子关于 GLM 模型上传数据、把用户 Git 上的数据上传上去的事情，就会让我想一想：它的原料到底是什么？

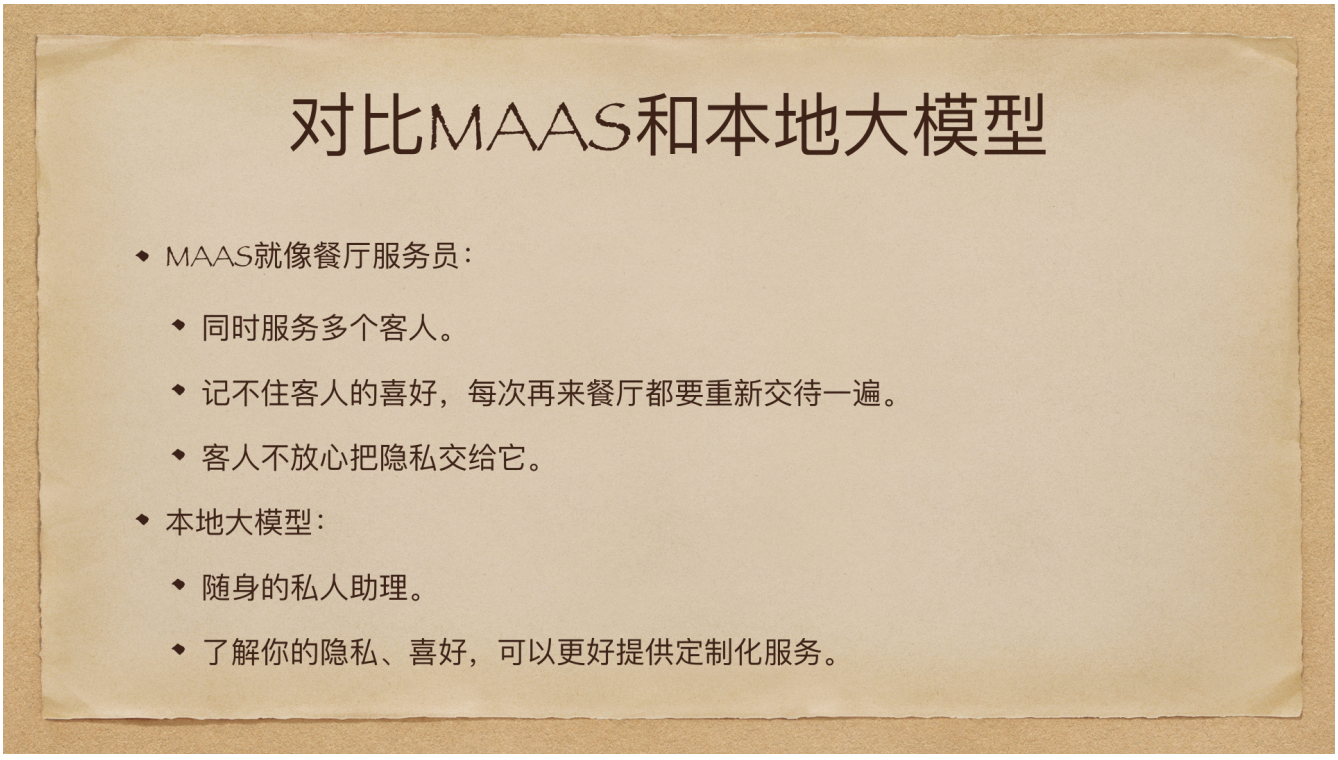
00:40:38 · codedump

很有可能是我的隐私数据。即便这里面没有财务信息，没有服务器密钥，我是不是愿意让自己的隐私数据变成它们的原料？这是我在想的问题，也是我觉得这个服务可以用本地大模型来做的原因。

00:41:11 · codedump

我甚至觉得，很多现在调用大模型 API 的服务，都可以再想一下：能不能把 API 大模型换成本地大模型，再针对本地模型做优化？也许这样的产品形态也能成立。还有一点，做本地大模型产品，可能天然就能避开一些大公司的竞争。

## 餐厅服务员与私人助理



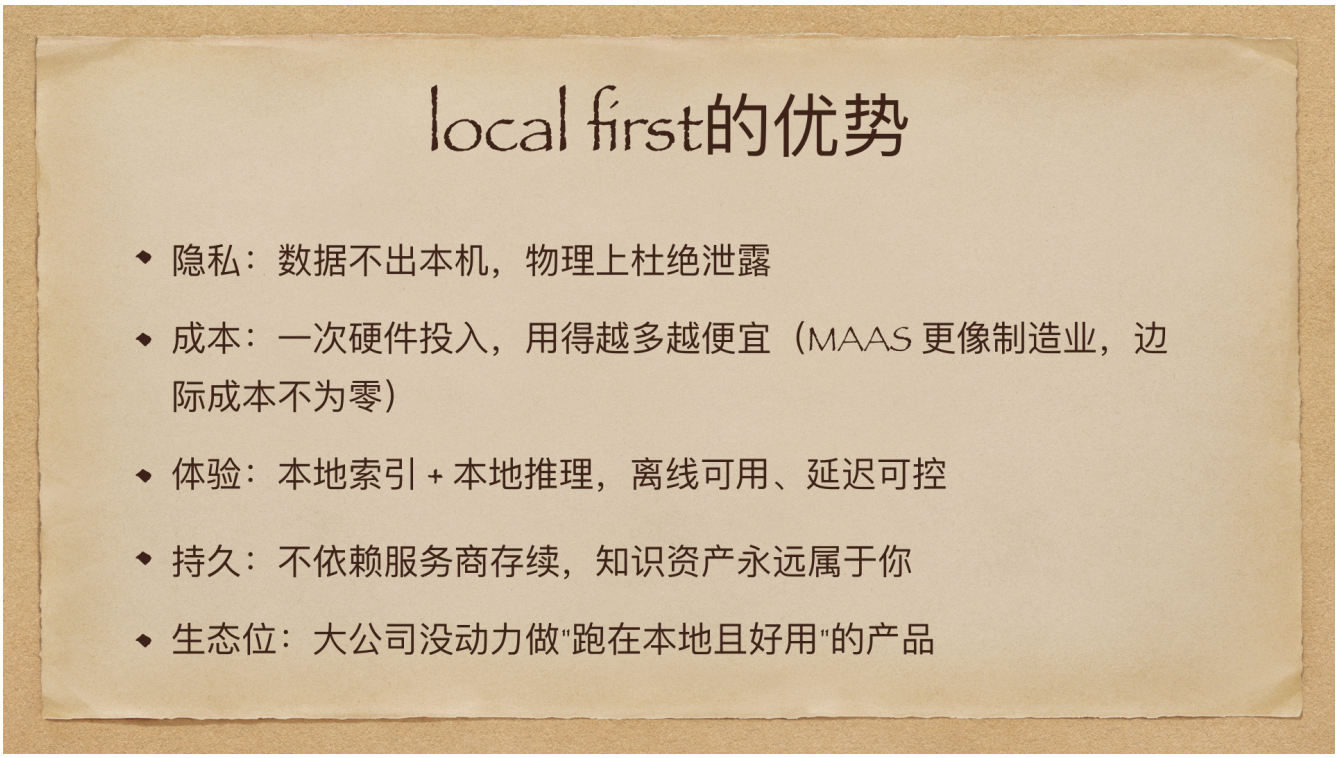

MaaS 与本地模型：比喻（讲者提供版幻灯片，第 16 页）

00:41:54 · codedump

因为在我看来，大公司的商业模式还是走 service 模式，不太会去做这种使用本地大模型的服务。再打一个比方：MaaS 更像餐厅服务员。餐厅可能很高级、很优雅，但服务员同时要服务多个客人。

00:42:28 · codedump

这里有几个问题。首先，他记不住每个客人的喜好。比如我吃素，不加葱，不加香菜；每次进同一家餐厅，哪怕遇到同一个服务员，可能还得重新交代，因为他没办法提供非常个性化的服务，也不会记住我的喜好。其次，客人不放心把隐私交给他，因为他毕竟是餐厅的服务员，不是由我控制的。

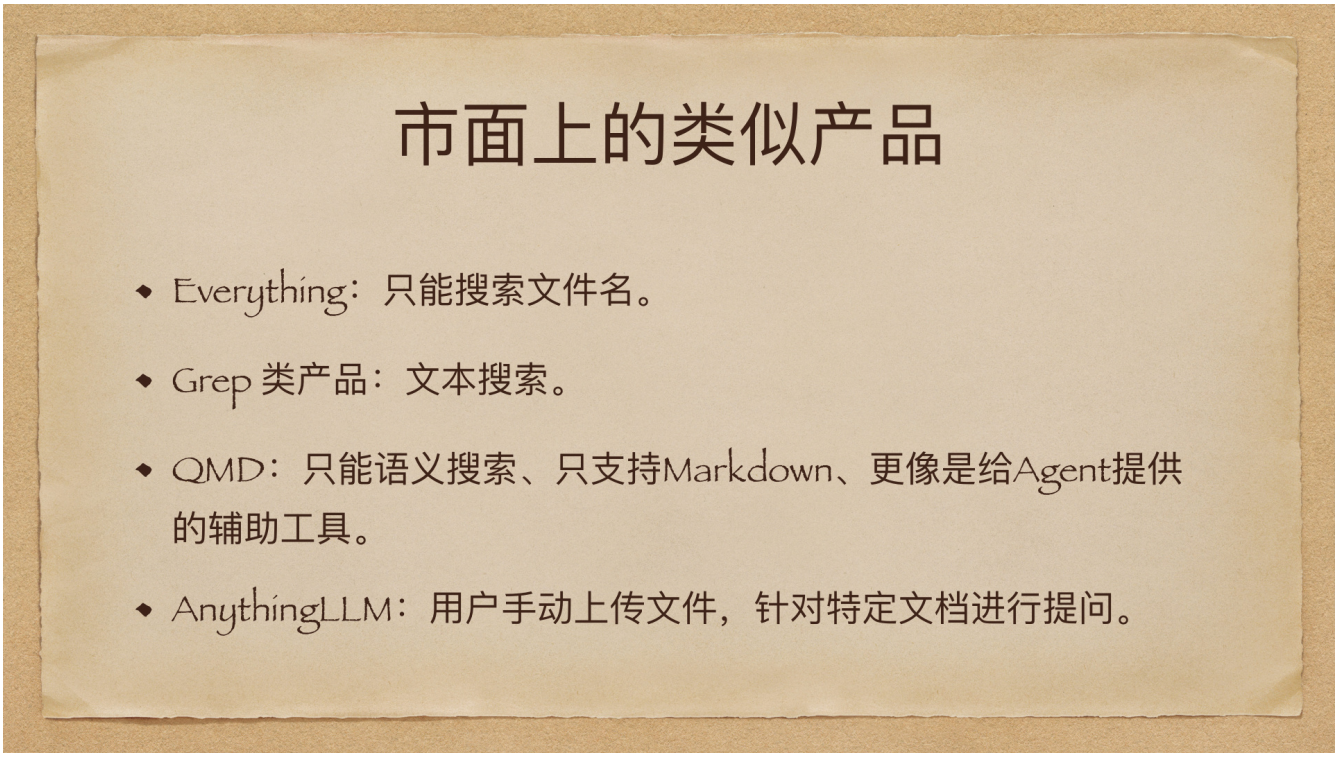



本地优先的优势（讲者提供版幻灯片，第 17 页）

00:43:01 · codedump

本地大模型则更像一个随身的私人助理。我可以放心把一些隐私和喜好告诉他，他也可以给我提供更多定制化服务。这是我对 MaaS 和本地大模型的一些想法。至于本地优先的优势，前面已经讲过很多，就不再重复了。

## 与类似产品的比较




类似产品（讲者提供版幻灯片，第 18 页）

00:43:33 · codedump

最后对比一下市面上类似的几个产品。Everything 是 Windows 上很出名的产品，我这里把它看作文件名搜索工具。另一类是 Unix 上 grep 这样的文本搜索工具。但还是前面那个问题：很多时候，用户并不知道要搜的文档里，具体关键词是什么。

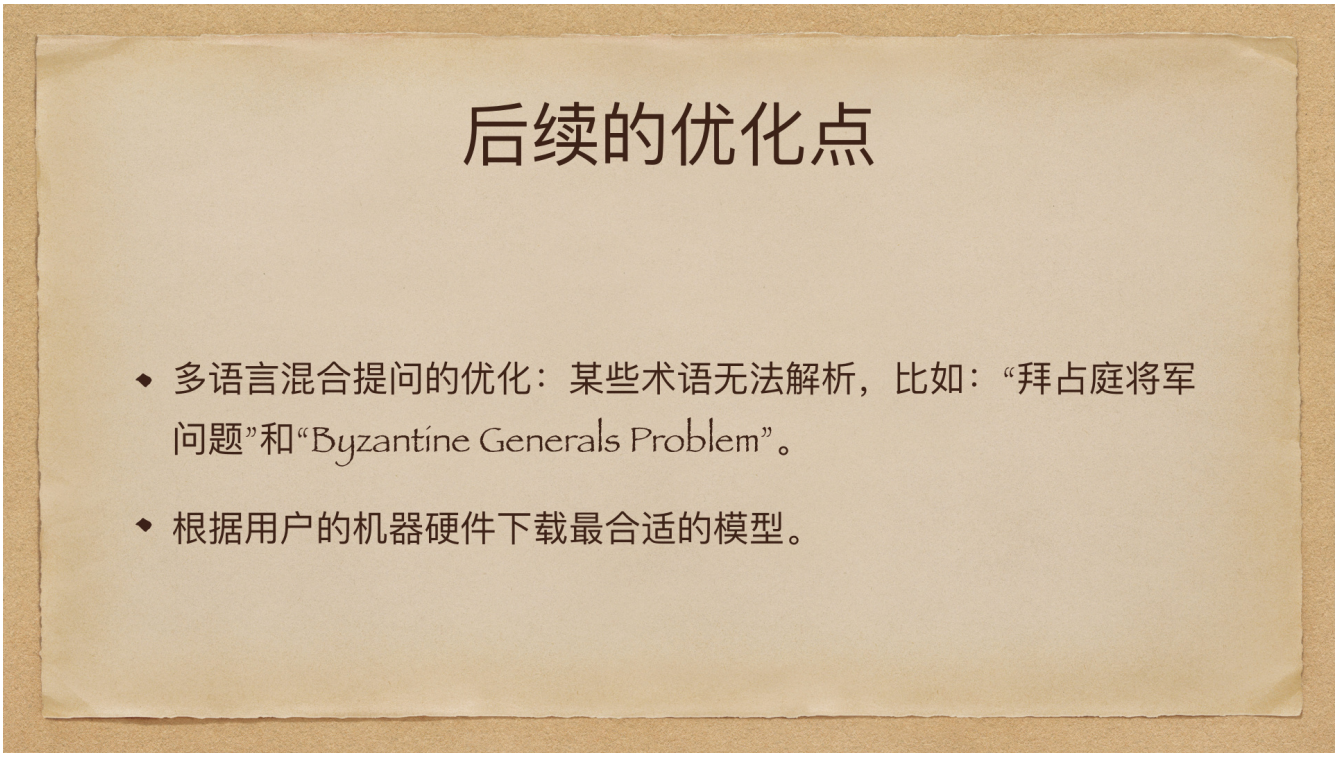
00:44:07 · codedump

所以这类产品与我想解决的问题不完全在一个方向上。比较像的还有 QMD，它是目前比较出名的开源项目。按我看到的情况，它主要做搜索，不负责回答；目前主要支持 Markdown。我看它的产品形态，更像是通过 MCP 服务提供给 agent 的辅助语义搜索工具。

00:44:40 · codedump

还有 AnythingLLM。我没有实际用过，只是在官网看了演示视频，感觉它需要用户手动上传一个 PDF 之类的文件，再针对这个文件提问。但回到刚才的场景，很多时候用户并不知道答案在哪份文档里，尤其是文档特别多的时候。

# 后续优化：跨语言查询与硬件适配




后续优化（讲者提供版幻灯片，第 19 页）

00:45:14 · codedump

要求用户手动上传文件，其实已经暗含了一个前提：用户知道问题在哪份文档里。而我想解决的问题不太一样。这就是我调研过的几个类似产品。最后讲后续优化。这个项目还很早期，有很多可以优化的地方，我再做一个演示。

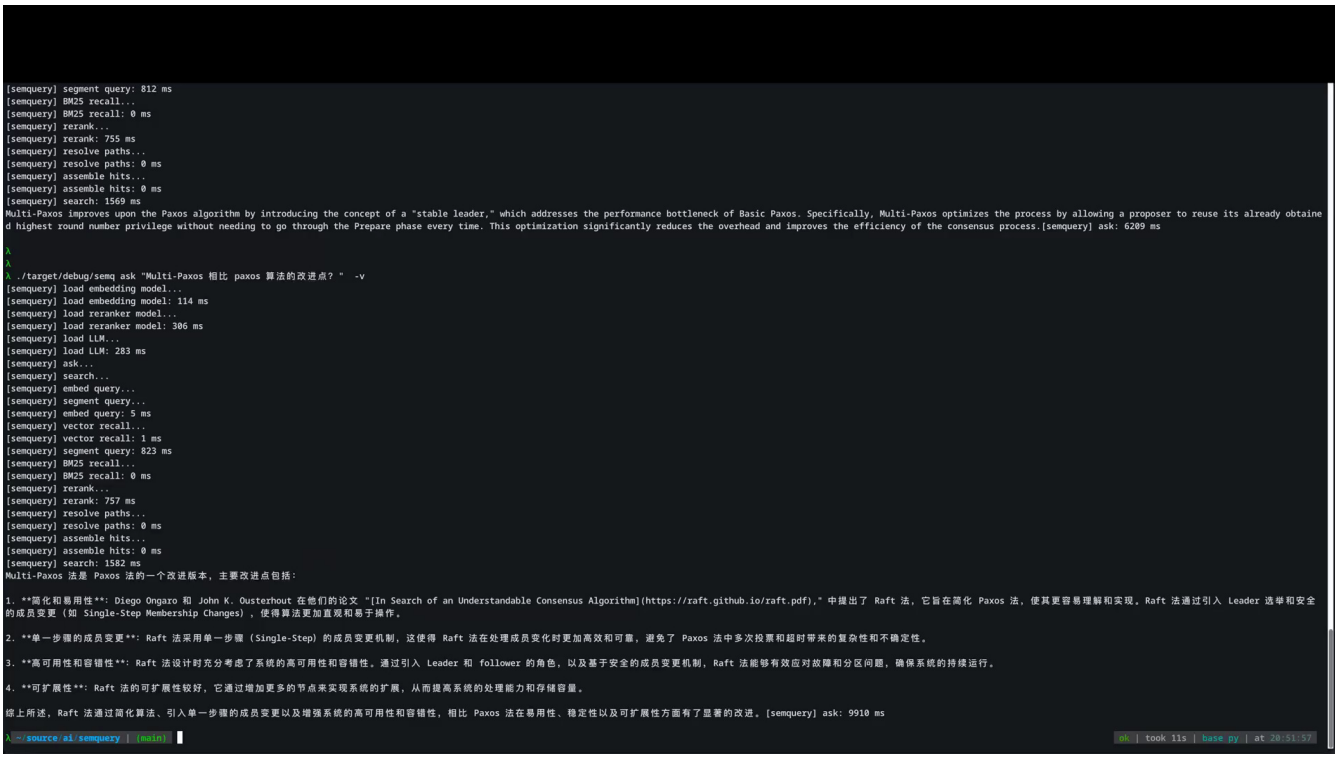
00:45:47 · codedump

刚才用 ask 命令，可以让它回答一个英文问题。如果把同一个问题换成中文表述，中文和英文的意思其实是一模一样的，我们来看一下。

00:46:27 · codedump

这会儿又能查到了。我下午试的时候，同样的问题，因为文档是英文的，用中文问，它回答不出来。所以我想表达的是：当文档是英文、问题是中文时，表现有时不是太好。

配图说明：下图展示的是现场模型生成的答案，用于记录演示过程，不能当作经校验的算法说明；讲者在这一段讨论的是跨语言提问的效果与稳定性。




现场演示：英文文档与中文提问（成片 46:50 截图）

00:46:59 · codedump

另外还有一些术语，它可能没办法解析。比如“拜占庭将军问题”，我试着问一下。

00:47:27 · codedump

这回又能回答出来了，看来还是不太稳定。我之前试过直接用中文提问“拜占庭将军问题”，整个文档是英文的，它没有理解这个术语，只有换成英文版本的问题才回答出来。我把英文的问题拷贝出来，再看一下。

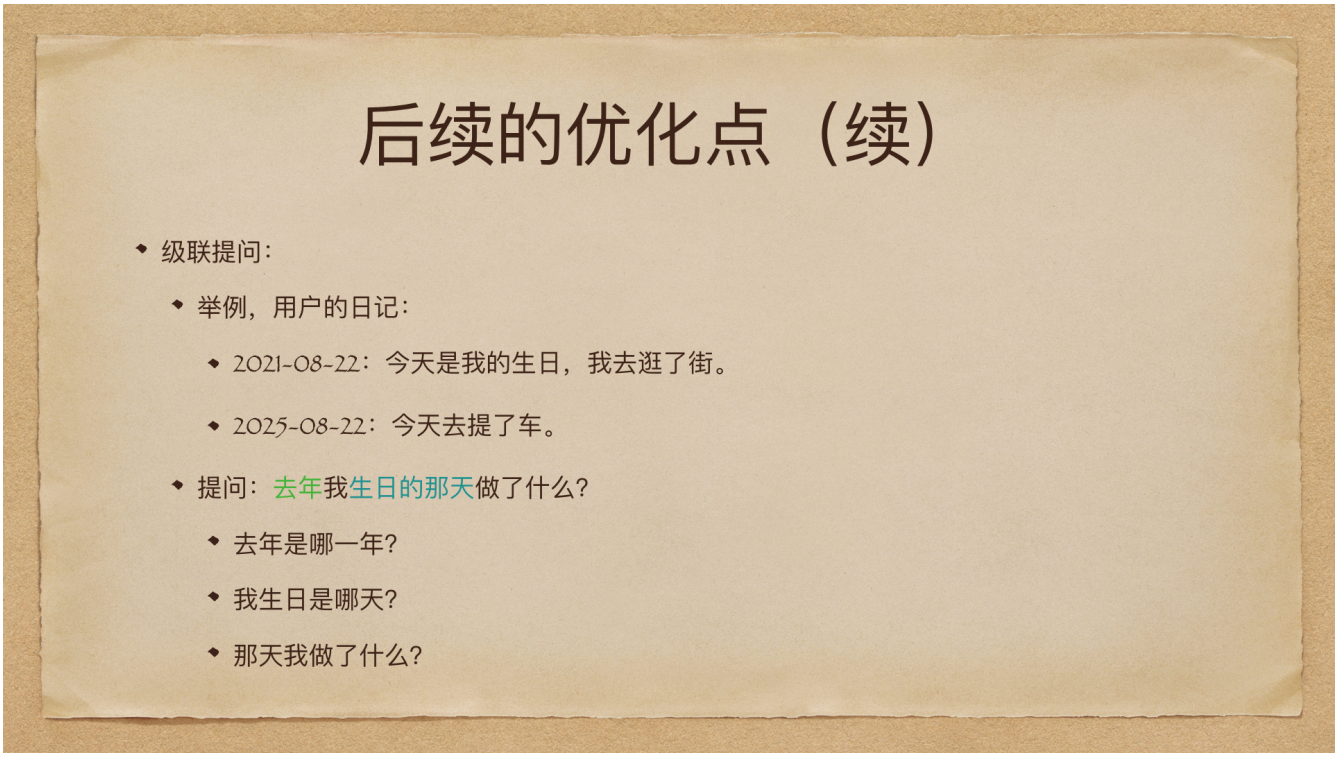
00:48:10 · codedump

在我之前的调试里，用中文提问“拜占庭将军问题”，它是回答不出来的。我不知道今天为什么又可以回答出来，这个还得再看。总之，表现还不太稳定：英文文档配中文提问，很多时候效果不太好，这是后面的一个优化点。第二个优化点是，目前 embedding、rerank 和 LLM 这几个模型，都写死在代码里。

00:48:57 · codedump

后面可能会加一个功能：判断用户的机器配置，再下载合适的模型。这是第二个优化点。前面两个还好，第三个我觉得更头疼：有些问题是级联提问。像刚才解释“拜占庭将军问题”，其实是比较简单、直接的问题。

## 级联问题：去年生日那天做了什么




级联提问（讲者提供版幻灯片，第 20 页）

00:49:30 · codedump

但假设用户有两条日记：2021 年 8 月 22 日写的是“今天是我的生日，我去逛了街”；2025 年 8 月 22 日也是生日，但那条日记里没写“生日”这两个字。现在用户问：“去年我生日那天做了什么？”这就是一个级联式的提问。

00:50:08 · codedump

我这里借用数据库里的一个说法，意思是某个查询依赖多个查询的结果。比如先从 A 表拿到一个结果，再和 B 表的结果结合，最后才能回答。在这个例子里，也得先回答几个问题：第一，“去年”是哪一年？只有理解了这个，才能找到 2025 年的日记。

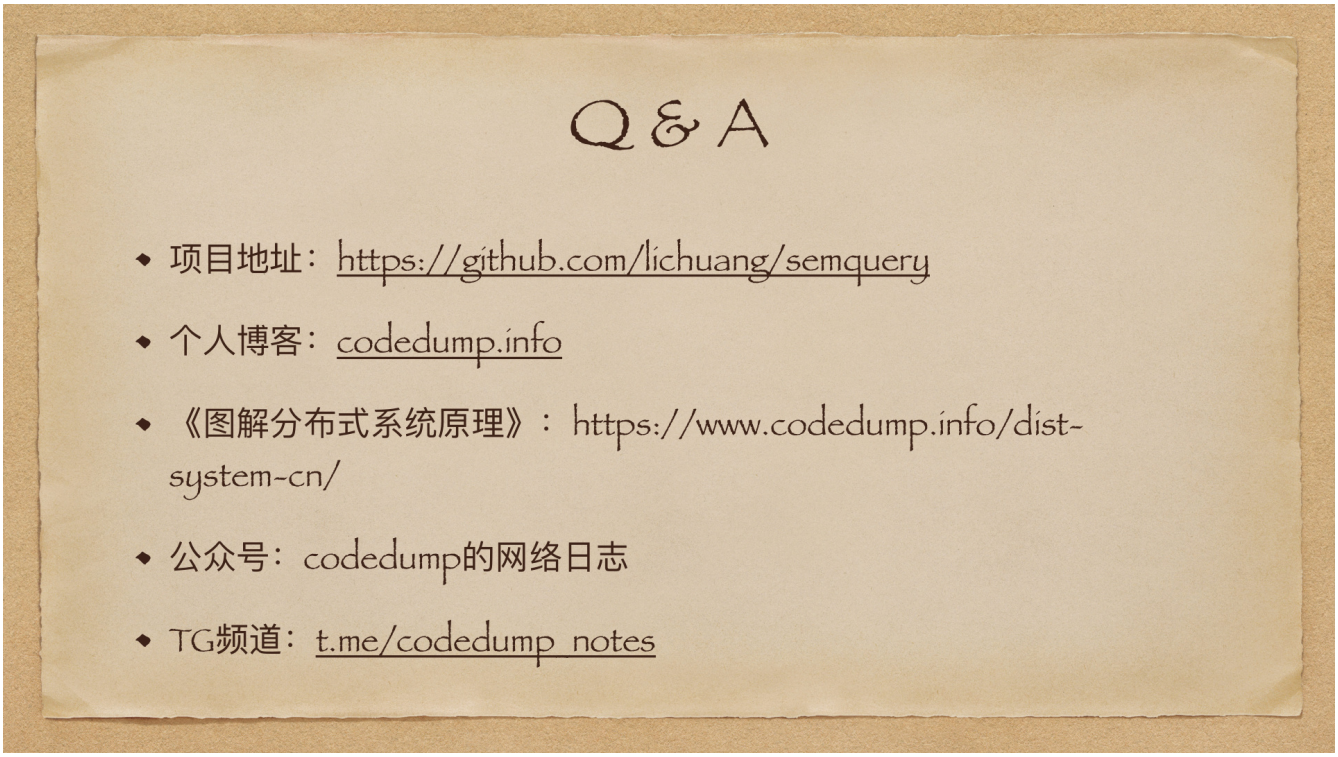
00:50:43 · codedump

第二，生日到底是哪一天？按目前的实现，还要先找到生日是 8 月 22 日，再把两个结果结合，才知道要找的是 2025 年 8 月 22 日那天的日记。根据前面讲的索引构造方法，目前我想不到太好的办法。

00:51:19 · codedump

当然，我还没有实际尝试，但我自己觉得，这种级联问题现在的效果应该会很不好。这些就是后续的优化点。今天大概就讲到这里，最后这一页是项目地址、个人博客、公众号和 Telegram 频道。我的分享就是这些。

# 问答：没有结果、重复文件与硬件要求




问答与项目链接（讲者提供版幻灯片，第 21 页）

00:51:49 · 交流者

好，大家有什么问题吗？

00:51:54 · 观众

李老师，我有几个问题。谢谢您的分享。第一个是错误处理的问题：如果索引的文档里，其实没有我想要找的内容，但我还是问了这个问题，这种错误反馈怎么处理？

00:52:17 · codedump

它会反馈说找不到。我今天还专门试过，随便取一个肯定不会在文档里的问题。这里我随便试一下。它会反馈 sorry，找不到相关内容。比如这里就会说，找不到王心凌相关的信息。

00:52:53 · 观众

第二个问题是重复文件。我的个人文件挺乱的，因为没有整理，会有很多重复文件。如果我偏偏要找的就是这些内容，遇到重复文件时，系统怎么处理？

00:53:11 · codedump

你指的重复，是计算机定义的那种，比如两个文件 MD5 相同，还是里面 90% 的内容相同？是内容完全一样的文件吗？

00:53:27 · 观众

对我而言，就是完完全全相同，只是放在不同的地方，文件名可能也不一样。

00:53:35 · codedump

这个可以在后面做优化。先算一下文件的 MD5，如果一样，就不重复加进去了。可以做这个优化。

00:53:50 · 观众

第三个问题，要达到比较好的使用效果，对电脑硬件的要求大概是什么样？

00:54:08 · codedump

我现在还没有测过，一方面没有更多电脑，另一方面，我们程序员买电脑时通常尽量选好一点的。后面可能会给我太太试一下。目前几个模型还是写死的，后面应该先判断硬件，再下载合适的模型。我的电脑也算比较好，还没有做更低配置的兼容性测试，所以最低硬件要求暂时回答不了。现在能说明的，就是刚才演示的情况：这台 2021 年、64 GB 内存的 MacBook Pro，面对刚才大约 18 万英文字符的测试文档，可以在七八秒内回答问题。

## 问答：模型内存与既有文件结构

00:55:12 · 观众

还有一个问题：搜索系统 idle 的时候，内置 LLM 的 offload 有没有优化？比如真正需要用的时候才加载，或者预测会有较长时间使用时再加载，其他时候释放内存。这方面有优化吗？

00:55:42 · codedump

目前没有。一启动就会加载进内存，然后常驻。你提了一个好问题，后面产品化时，我会把这个事情考虑进去。

00:55:59 · 观众

还有一种情况：一个人在不同人生阶段，文档的整理状态可能不同。某个阶段，他的文档 entropy 很低，规划得很完整，目录和文件都有规则；后来又开始乱放。那么以前这些低 entropy、已经整理好的结构，能不能被利用？

00:56:31 · codedump

你指的是目录里又有新文件进来、新目录出现，或者文件移出去这种变化吗？

00:56:39 · 观众

不是。我的意思是，他以前很爱规划，目录摆放很有规则，按那个结构就能很好地索引。但后来可能把文件都直接丢在 MacBook 桌面，不再整理。以前整理过的那些文件，已有的整理成果有没有可能复用？刚才评论区提到了别的问题——我这里说的不是级联问题，而是文件原本已经被妥善整理的这部分。

00:57:29 · codedump

目前我只会加载用户指定的目录。像刚才的 add 命令，用户让我索引哪个目录的文件，我就索引哪个目录。这个目录乱不乱、摆放整不整齐，不在产品关注的范围内。我只关注你指定目录里的文件，文件怎么摆放没有太多关联。

## 问答：企业部署与多个并发问题

00:58:14 · 观众

最后一个问题，您刚才提到了律师事务所。他们公司可能会共享比较大的文件，需求可能是做一个内部服务器。这种并发问题有考虑吗，后面会优化吗？

00:58:38 · codedump

目前这个产品面向个人，还不是面向多人的。如果做成 To B 的事情，首先可能就需要一个团队，包括销售、市场团队。现阶段我还想不了那么远，想的是尽早把产品做出来，让大家先用上。至于到公司里部署一台服务器，目前还不在计划里。

00:59:21 · 观众

那个人连续提问怎么处理？是先暂停，还是对一个人同时问多个问题，有相应的处理方式？

00:59:37 · codedump

是指一个人同时问好几个问题吗？

00:59:40 · 观众

对，可能 UI 做出来后会更好说明这个场景。

00:59:45 · codedump

明白，这也可以优化。如果确实有这种需求，比如同时提问几个问题，再切换不同窗口、不同对话框看结果，应该可以支持，不是特别复杂的事情。

01:00:03 · 观众

好，谢谢李老师。

## 问答：多人隐私搜索与关系图

01:00:09 · 交流者

大家还有问题吗？我有一个问题：如果考虑多人协作，但又是隐私搜索的场景，你有看到比较好的解决方案吗？

01:00:30 · codedump

多人，是指两个人看的都是同一份文档吗？

01:00:36 · 交流者

是我有一套文档体系，你也有一套。比如律师协作时，你有一系列卷宗文件，我也有一系列卷宗文件。

01:00:50 · codedump

目前还没有考虑这种多人协作的情况。现在主要的场景，还是一个用户下载 App，针对本地目录建立索引、搜索和提问，暂时没有跨机器的功能。这个产品后面会进化成什么形态，很大程度上要看放出去后得到的反馈。

01:01:25 · codedump

如果这个需求确实很大、呼声很高，可能会考虑。但就目前而言，还没有想得这么远。

01:01:42 · 交流者

我提这个问题，是想请你先大概设想一下：这种多人、又具有隐私搜索性质的场景，可能做到什么程度？

01:01:53 · codedump

比如文档在你的机器上，但我可以在我的机器上对你本地的文档提问，是这个意思吗？

01:02:08 · 交流者

我也不确定边界。也许只是一个匹配：让我知道你有一份与问题相关的重要文档，能够推荐就好。这样我就可以找你。

01:02:18 · codedump

我初步想，是不是每个人要开放一个网络端口，共享给别人提问，有点像 MCP 的意思。可能要这样做。

01:02:36 · 交流者

第二个问题是级联查询的方案。好像不得不引入一些关系图，再让 agent 去维护这些关系的结构。

01:02:45 · codedump

有可能。我现在还没有想得特别好，这也是前面几个问题中，我觉得最复杂的一个。可能整个数据体系都得改。

01:03:01 · 交流者

它已经是另一种结构了，涉及符号之间的语义推理，不只是向量化，感觉已经换了一套思路。

01:03:11 · codedump

对，是的。

## 结束交流

01:03:15 · 交流者

好，我没有其他问题了。

01:03:19 · codedump

好的，感谢。

01:03:24 · 交流者

看看其他小伙伴还有没有问题，或者想交流的内容。好像没有，那我们今天就结束了。

01:03:54 · codedump

好，谢谢大家。

01:03:56 · 交流者

谢谢。

01:03:58 · codedump

那今天就这样，谢谢大家，拜拜。

01:04:05 · 交流者

非常感谢，拜拜。

项目：https://github.com/lichuang/semquery | 讲者博客：https://www.codedump.info/