Week01｜Lesson03｜别一上来就写 RAG：企业级 AI 到底该走哪条数据工程路线

把路线先于模型技巧这件事讲透，让学生第一次看见企业级 AI 的真实工程展开图

本讲结构

别一上来就写 RAG：企业级 AI 到底该走哪条数据工程路线

这一讲不比谁的 Prompt 更会写，也不先讲哪种检索更高级。它只做一件事：把你从“功能能演示”拉到“路线能上线”。

上一讲 返回 Week01 下一讲

下载讲义

提供适合离线阅读的 PDF 版和适合批注整理的 Word 版。

PDF 版 · 打印 / 离线阅读 Word 版 · 批注 / 二次整理

> ! 一句话判断
>
> 企业 AI 的主问题从来不是“模型够不够聪明”，而是你有没有走对从数据、边界、工具到治理的那条路线。

这节课先建立三个基本判断

路线判断

先判断系统该停在 PoC、试点，还是必须往平台化路线升级，而不是一开始就沉迷局部技巧。

项目定位

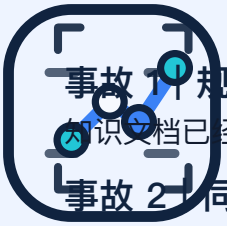
OmniSupport Copilot 还是贯穿案例，但它只是验证器。课程主线始终是企业 AI 数据工程能力。

后续 14 周

后面每一周不是堆技术点，而是在补一条从输入、检索、动作到治理的上线链路。

事故开场：Demo 能答，上线就炸

先别急着讲术语。先看一个已经在演示现场“答得不错”的系统，为什么一接真实业务就爆雷。



事故 1｜规则更新了，答案还在讲旧版本

知识文档已经更新，但索引没重建，系统继续把旧政策说得很自信。

事故 2｜同名术语跨部门口径不一致

“冻结”“挂起”“限制”在客服、风控、运营里的定义不一样，系统却当成一个意思回答。

事故 3｜低权限用户看到了高敏 SOP

你以为自己在做问答，实际上已经碰到了权限分层、PII 和动作边界。

事故 4｜坏例出现后根本没法复现

回答错了，但你说不清是解析、切片、召回、重排还是 Prompt 变了。

> ⚠ 这不是模型太弱，而是路线选错了
>
> 如果你从一开始就把企业 AI 理解成“Prompt + 向量库 + LLM”，你迟早会在真实数据、权限、工具调用和治理环节撞墙。

这节课只回答三个问题

撞墙

问题 1

为什么脚本式 RAG 可以快速验证想法，但迟早会在生产里撞墙?

失控

问题 2

企业级 AI 系统至少由哪些层组成，为什么不能只盯检索和生成?

返工

问题 3

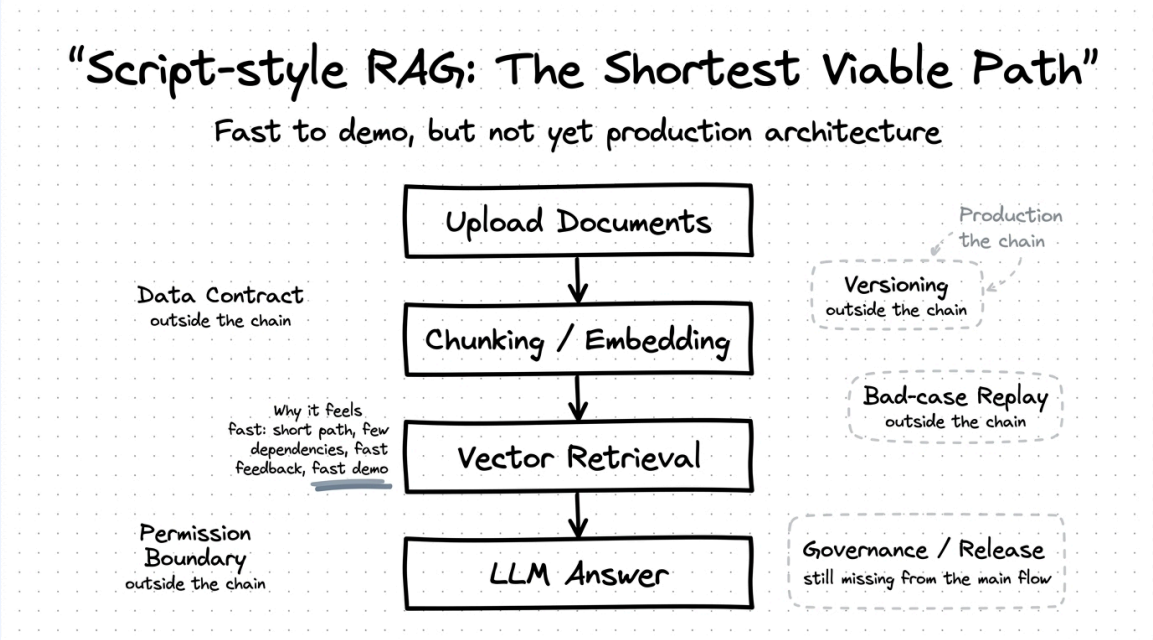
后续 14 周为什么不是过度工程化，而是在补上线级 AI 的必要能力?

> ℹ Note
>
> 今天不堆术语，也不提前透支后面所有实现细节。这节课要先把“路线判断”立住，后面的技术才不会学成零散工具箱。

为什么大家总会先写脚本式 RAG

先承认一件事：大多数团队第一反应写脚本式 RAG，并不荒唐。 因为从“想法”到“可演示结果”，这确实是最短路径。

最短可用路径



这张图真正要表达的是：脚本式 RAG 之所以快，不是因为它已经完整，而是因为很多上线能力还留在链外。

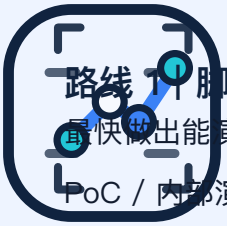
为什么它会成为团队直觉

- 依赖少，几天就能做出演示
- 业务方很容易看到“它会答”
- 团队会误以为“离上线已经不远”
- 真正昂贵的治理工作被延迟支付

> **Note**
>
> 脚本式 RAG 的问题不是“不能做”，而是很多团队把它误当成了正式路线。

三条路线总览：你不是在选技术栈，而是在选后果

下面这三条路线没有谁天然更高级。 真正重要的是：什么阶段能停，什么信号一出现就必须升级路线。



路线 1｜脚本式 RAG

最快做出能演示的链路。适合 PoC，不适合把复杂度继续拖到上线后。

PoC / 内部演示

路线 2｜服务化 RAG

比脚本更稳定，有 API、有缓存、有日志，但还远没到平台化治理。

试点 / 小范围开放

路线 3｜企业级 AI 数据工程平台

目标不是“一上来做大平台”，而是清楚知道最终必须补齐哪些层。

正式上线 / 多团队协作

脚本式 RAG：快，是因为你绕开了最贵的问题

- 你今天能快，是因为数据契约、权限边界、回放能力和发布治理还没真正进场。
- 它最适合回答“这件事值不值得做”，不适合回答“这套系统能不能上线”。
- 一旦进入真实用户、真实权限、真实动作链，复杂度会集中爆发。

该停在这里 PoC、概念验证、演示链路。

必须切路线的信号 出现真实用户、坏例回放、跨部门口径、权限分层、工具动作。

服务化 RAG：你已经更稳了，但底层资产仍然可能很乱

- API、缓存、日志让系统比脚本更像服务，但这不等于你已经有了统一数据底座。
- 你可以支撑试点，却未必能支撑多团队协作、持续回归和版本治理。
- 它解决的是应用层稳定性，不自动解决数据资产、证据边界和治理闭环。

该停在这里 小范围试点、受控用户群、有限动作范围。

必须切路线的信号 开始需要统一契约、统一评测、统一回滚和统一发布口径。

企业级平台路线：目标不是复杂，而是可控

- 真正上线后的贵，不是首版功能，而是跨团队维护、事故定位、回滚与责任边界。
- 平台化路线把这些能力前置成一条工程链，而不是继续让每次事故都临时补洞。
- 它不是让你 Day 1 做完全部，而是让你 Day 1 就知道最终要补齐到哪里。

核心价值 可追溯、可观测、可治理、可持续迭代。

最容易误解的点 不是“大而全”，而是“方向一开始就别走偏”。

脚本式 RAG 为什么快，为什么两周后就开始脆

快在什么地方

- 路径短、依赖少，反馈快
- 先用最少的数据工程把“会答”做出来
- 现场 Demo 看上去通常会很顺
- 对团队来说，启动成本极低

脆在什么地方

- 复杂度被延迟支付，问题集中在上线前爆发
- 版本、权限、坏例回放、发布边界都不清楚
- 一旦数据换批、规则更新、工具链变化，就很难复盘
- 你以为自己缺模型技巧，实际上缺的是工程链

> ! 关键判断
>
> 脚本式 RAG 的最大风险，不是效果差，而是它会让团队误以为“剩下只是再调一调模型”。

服务化 RAG 已经更进一步，但为什么还不够

它已经解决的事

- 把脚本收成服务
- 有接口、有缓存、有基本日志
- 更容易支持一小批真实用户

它还没自动解决的事

- 数据是否有统一底座和版本语义
- 索引是不是资产，而不是一堆临时产物
- 评测、回滚、发布是否可绑定

最常见误判

“我们已经 API 化了，所以已经工程化了。”

真实情况是：你只是把应用层做厚了，底层资产和治理未必跟上。

> ⚠ 不是 API 化就等于工程化
>
> 服务化 RAG 解决的是调用稳定性，不自动解决数据契约、证据边界、动作分级和治理闭环。

真正的目标对象：不是一个 RAG 系统，而是一条企业 AI 数据工程链

数据 把输入做成可追溯资产，而不是临时喂给模型的原料。

检索与生成 让回答不仅像样，还要带证据、能回归、可校验。

工具与行为 让系统从“会答”进入“能办”，但动作权限必须前置设计。

治理与发布 把评测、Tracing、版本、回滚和责任边界收成上线能力。

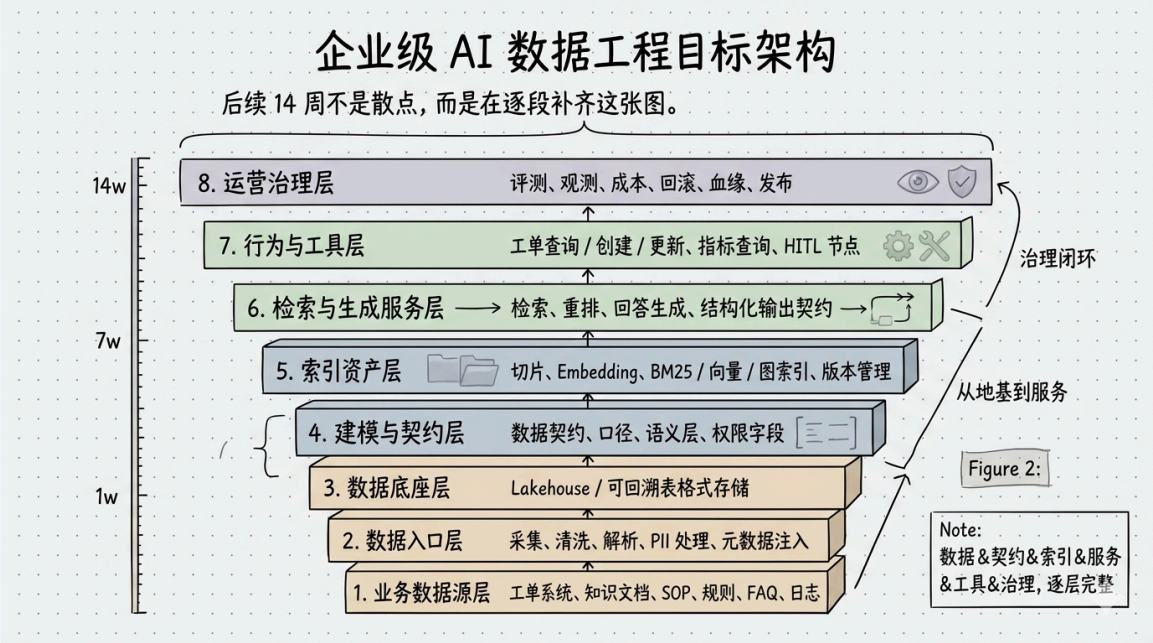


Figure 1

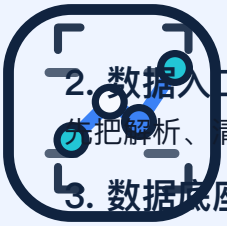
> **Note**
>
> 这张图不是装饰。以后遇到任何问题，都回到这里问三句。 我们卡在哪一层？我们缺的是能力还是顺序？我们是在补模型技巧，还是在补上线链路？

八层目标架构：缺哪层，就会在哪种事故上翻车

点击下面任意一层，看它到底解决什么事故、交付什么能力、最小抓手是什么。

1. 数据源层

知道答案到底来自哪一批业务资产。



2. 数据入口层

先把解析、清洗、PII 和元数据做对。

3. 数据底座层

让回溯、回放、回滚有地方落。

4. 契约语义层

统一口径、字段语义和权限边界。

5. 索引资产层

让检索不是一个黑盒库，而是一类可版本化资产。

6. 检索生成层

让回答带证据、可校验、可重放。

7. 行为工具层

从“会答”进入“能办”，同时守住动作边界。

8. 治理发布层

把评测、Tracing、版本、回滚和责任归属连起来。

1. 数据源层

- **缺了会怎样：** 你根本说不清系统到底建立在什么数据之上，后面也谈不上 owner、版本和责任。
- **最小抓手：** 数据源清单、owner、刷新频率、使用边界。
- **在 OmniSupport 里：** FAQ、SOP、工单系统、规则文档先要登记成可管理资产。

2. 数据入口层

- **缺了会怎样：** 文档解析错了、PII 没处理、元数据没注入，后面所有链路都会带着脏输入运行。
- **最小抓手：** source_fingerprint、pii_redaction_flag、parse_quality_score。
- **在 OmniSupport 里：** 同一份业务规则从 PDF、邮件和后台导出进入系统时，要能留下一致的进入记录。

3. 数据底座层

- **缺了会怎样：** 出事故后你没有地方做 time travel、回放和回滚，只能靠猜。
- **最小抓手：** snapshot_id、ingest_batch_id、可回溯存储。
- **在 OmniSupport 里：** 新旧业务规则、FAQ 和案例库要能按快照回看，而不是互相覆盖。

4. 契约语义层

- **缺了会怎样：** 同名术语在不同部门口径不一致，权限字段也没人真正定义。

- **最小抓手：** 契约版本、标准术语表、访问等级字段。
- **在 OmniSupport 里：** “冻结”“限制”“挂起”必须先统一语义，再谈检索和回答。

5. 索引资产层

- **缺了会怎样：** 你只有一个“向量库”，却不知道切片策略、Embedding 版本和索引发布时间。
- **最小抓手：** chunk_policy_id、embedding_version、index_release_id。
- **在 OmniSupport 里：** FAQ、SOP、工单经验不能全塞进一个黑盒索引里，要按资产分类管理。

6. 检索生成层

- **缺了会怎样：** 结果看起来会答，但证据不稳、结构不可验、坏例难复现。
- **最小抓手：** 检索 trace、重排理由、输出 schema 校验。
- **在 OmniSupport 里：** 回答不仅要给结论，还要能说清引用了什么证据、为什么这么说。

7. 行为工具层

- **缺了会怎样：** 系统只能答，不能办；或者更糟，答完就越权去办。
- **最小抓手：** 工具契约、动作分级、HITL 节点、幂等约束。
- **在 OmniSupport 里：** “建议建单”和“自动建单”必须分层，不是一个开关。

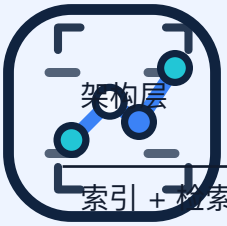
8. 治理发布层

- **缺了会怎样：** 评测、Tracing、版本、发布记录互相断开，事故一来没人说得清该回退哪里。
- **最小抓手：** release_id、eval_run_id、trace_id、回滚 Runbook。
- **在 OmniSupport 里：** 每次上线都要能绑定到具体评测基线、索引版本和工具策略。

把 OmniSupport Copilot 映射回这张总图

项目是验证器，不是课程主线。但正因为它贯穿始终，它最适合拿来说明：为什么这不是“做个问答机器人”这么简单。

架构层	在客服 Copilot 里具体承担什么	如果这一层缺了，最先爆什么问题
数据源 + 入口	接 FAQ、SOP、业务规则、工单数据，并完成解析、清洗、PII 处理	规则更新不一致，解析错误直接进索引
底座 + 契约	保留快照、统一术语、定义权限字段和业务口径	同名术语冲突、旧版本无法回看、责任边界不清



架构层	在客服 Copilot 里具体承担什么	如果这一层缺了，最先爆什么问题
索引 + 检索生成	管理 FAQ / SOP / 工单经验索引，输出带证据的结构化回答	召回不可解释、证据不稳定、坏例难复现
工具行为	查询工单、建议建单、升级人工、审批节点	一旦动作边界不清，就会从“会答”滑向“越权执行”
治理发布	评测、Tracing、变更记录、回滚和 Runbook	事故无法定位，试点永远不敢真正上线

> ! 项目是验证器，能力是主线
>
> OmniSupport Copilot 的价值，不是把所有内容都绑进一个项目，而是让你每学一层，都能在同一个业务世界观里验证它到底为什么必要。

三阶段演进：MVP / 可试点 / 可上线

阶段 1

MVP

先证明业务问题值得做，但别假装自己已经接近上线。

必须具备

- 最小检索链路
- 最小证据引用
- 最小日志与坏例记录

故意先不做

- 不急着做自动工具执行
- 不假装已经有完整治理闭环

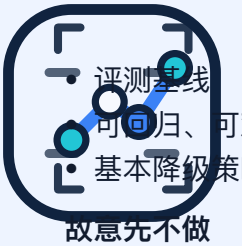
阶段 2

可试点

让小范围真实用户开始使用，但要先知道哪些能力已经必须进场。

必须具备

- 权限过滤



- 评测基线
- 可回归、可观测
- 基本降级策略

故意先不做

- 不急着做大而全治理平台
- 不为了规模幻觉提前堆重系统

阶段 3

可上线

纳入正式业务流程，开始真正承担发布、回滚和审计责任。

必须具备

- 版本、回滚、Runbook
- 评测与发布绑定
- 动作分级与 HITL
- SLO、Tracing、责任边界

故意先不做

- 不再依赖单人经验兜底
- 不把关键能力留给临场补洞

> ℹ Note
>
> 平台化不是要求你第一天就做全套，而是要求你第一天就别把方向走反。

Week02—Week15 不是加料，而是在补链

点下面任一周次，看看它在补总图里的哪一层，以及为什么这个顺序成立。

Week02

Week03

Week04

Week05

Week06

Week07

Week08

Week02 | 输入确定性、门禁与边界

先守住输入端。因为如果入口就不稳，后面的索引、评测和治理全部建立在漂移数据上。

Week03 | 采集与入湖

把数据从“临时拿来喂模型”升级成“可追溯进入底座的资产”。

Week04 | 快照与演进

让时间维度进入系统，后面才谈得上 time travel、回放和回滚。

Week05 | 口径与语义层

统一术语、字段和契约，不然后面检索再强也会答在错误口径上。

Week06 | 资产化编排

从 task-thinking 走向 asset-thinking，让数据工程路线能稳定地被编排和复用。

Week07 | 文档工程

企业文档不是“纯文本切片”就够，版面、表格、证据回指都会影响后续检索质量。

Week08 | 检索与生成一体化

这时才真正进入 RAG 核心层，但它已经建立在前面几层的输入和资产稳定性之上。

Week09–10 | Skills、工具与 Agent 行为边界

从“会答”走到“能办”，但前提是工具契约、权限分级和监督节点全部进场。

Week11–12 | 评测、Tracing、治理

把好坏判断、可观测性和发布责任正式收束成上线能力，而不是留给事后补救。

Week13–15 | GraphRAG、发布与上线收官

最后几周不是“再加料”，而是在已有总图上把复杂关系、发布和收官能力补齐。

课堂收束：后面不是过度工程化，而是在补上线级 AI 的必要能力

结论 1

脚本式 RAG 适合验证想法，但不适合作为企业级 AI 的正式路线。

结论 2

企业级 AI 的目标对象不是“一个向量库”，而是一条从数据到治理的工程链。

结论 3

后面 14 周不是堆技术点，而是在沿着同一张总图逐层补齐上线能力。

! 最终判断

如果一套 AI 系统只能回答“它现在能不能演示”，却回答不了“它将来怎么回滚、怎么评测、怎么审计、怎么维护”，那它就还没有真正走上企业级路线。

课后判断题

1. 如果你今天的系统还是“上传文档 → 切片 → 检索 → 回答”，它最可能先撞哪一堵墙：版本、权限、动作边界，还是坏例回放？
2. 在你的项目里，最该优先补的是哪一层：数据入口、底座、契约、索引、行为，还是治理？
3. 你们团队今天讨论的是“怎么上线”，还是还停留在“怎么演示”？