Week01｜Lesson04｜别急着写 RAG：项目骨架、风险边界和落地蓝图，为什么第一周就得定？

把“能演示”收口成“能协作、能验收、能负责”的工程起点

本讲结构

别急着写 RAG：项目骨架、风险边界和落地蓝图，为什么第一周就得定？

这一讲不再问“仓库能不能先跑起来”，而是先回答：一个企业级 AI 项目，凭什么有资格开工。Week01 要先把工程基线、风险边界和落地蓝图定下来，后面 14 周的实现才不会变成边做边返工。

上一讲 返回 Week01 进入实验

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

> ! 一句话结论
>
> 企业级 AI 项目最怕的，不是第一天效果差，而是第一天就没有统一工程基线。没有这条基线，后面每一周做得越多，返工越大。

冲突开场：E1042 为什么不是一个“回答”问题

Northstar Systems 的企业客户在升级 Edge Gateway 时失败，界面报错：

```
E1042
```

客户在支持助手里提问：

> Edge Gateway 升级失败，报错 E1042，我能不能直接回滚？

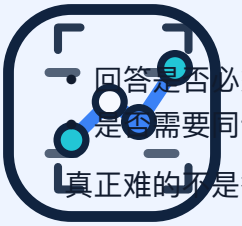
Demo 级系统通常会怎么做

- 去 PDF 手册里检索 E1042、rollback、gateway upgrade
- 拼出一段看起来合理的处理建议
- 回答“建议先回滚，再检查依赖版本”
- 现场演示时，这个答案通常已经够像“会做事”

它已经像一个答案，但还不像一个能负责的系统。

生产级系统还必须继续判断什么

- 当前客户的 SLA 等级和订阅权限是什么
- 这个产品线是否允许自动触发工单或回滚动作
- “回滚”是否属于高风险动作，是否必须进入 HITL



- 回答是否必须带证据锚点、trace_id 和 release_id
- 是否需要同步写入审计日志，供后续复盘和追责

真正难的不是答，而是判断系统有没有权利继续做后续动作。

> ⚠ 这节课真正要解决的问题
>
> 企业 AI 的难点不在回答，而在负责。Week01 要做的，不是先把功能写多，而是先把系统能负责到什么程度写清楚。

先看项目全貌：OmniSupport Copilot 到底是什么

OmniSupport Copilot 不是一门课里临时举例的小 Demo，而是这 15 周训练营贯穿到底的唯一工程基线。

更准确地说，它是一套面向 Northstar Systems 的准生产级多模态企业支持系统：既要支持知识问答，也要支持证据引用、工单联动、人工介入、评测、Tracing、版本与回滚。

项目定义：它到底要交付什么

- 面向企业支持场景，而不是开放域聊天
- 支持文档问答、证据引用、工单查询 / 创建 / 更新、指标查询
- 系统默认带 HITL、审计、回归评测、版本与回滚意识
- Week01 就要先立统一蓝图、工程基线、风险边界和验收口径

这不是它要做的

- 不做脱离业务边界的“万能助手”
- 不做无限制自动执行代理
- 不把工程化留到后面补

业务世界观：它为什么不是只靠 PDF 问答

- Northstar Workspace：帮助中心、FAQ、Release Notes、API 文档、工单
- Northstar Edge Gateway：安装手册、规格说明、接线图、故障排查视频
- Northstar Studio：教学视频、录屏教程、错误码手册、社区问答
- 数据形态天然是多模态：文档、ticket、音频转写、视频切片、截图与指标

这也是为什么课程后面会继续讲 contract、ingest、Iceberg、retrieval、tool、governance，而不是停在“PDF + 向量库 + LLM”。

仓库与实施原则

- GitHub 仓库：dataPro-lgtm/omnisupport-copilot
- 仓库骨架围绕 infra / services / pipelines / contracts / data / observability / evals / docs / runbooks

- Week01 默认走 Docker-only：先不要求学员本机预装 Python、PostgreSQL、MinIO
- 默认 compose 不把 PostgreSQL 暴露到宿主机 5432，避免和本机数据库冲突
- ANTHROPIC_API_KEY 在 Week01 可选：先把工程基线拉起来，再接真实模型
- 推进原则不是“先写功能”，而是 Data-first、Workflow-first、Evidence-first、Release-aware、Dual-scale
- 课程交付也不是单一标准，而是同时满足 Student Core Pack 本地可跑和 Instructor Scale Pack 规模演示

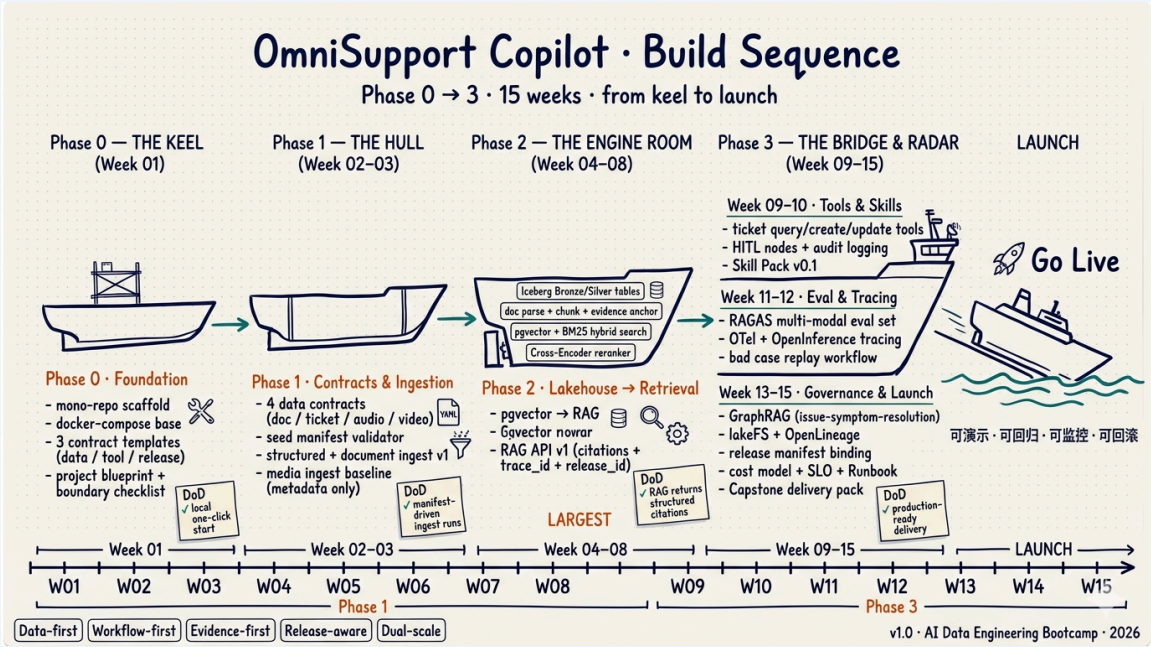


Figure 1

> **i** 把 Week01 放进整条路线里看
>
> 今天这节课要完成的，不只是“理解项目该怎么搭”。它真正对应的是整条建造序列里的 Phase 0 - The Keel：先把 mono-repo、compose 基线、contract 模板、项目蓝图和边界清单立住，后面 14 周才不是在沙地上继续往上堆。

为什么 Week01 就要先定工程基线

工程基线不是“先把仓库搭起来”的礼节动作，而是后面 14 周所有实现的共同边界。只要第一周不先把它定下来，返工会沿着一条非常固定的链条往后传。

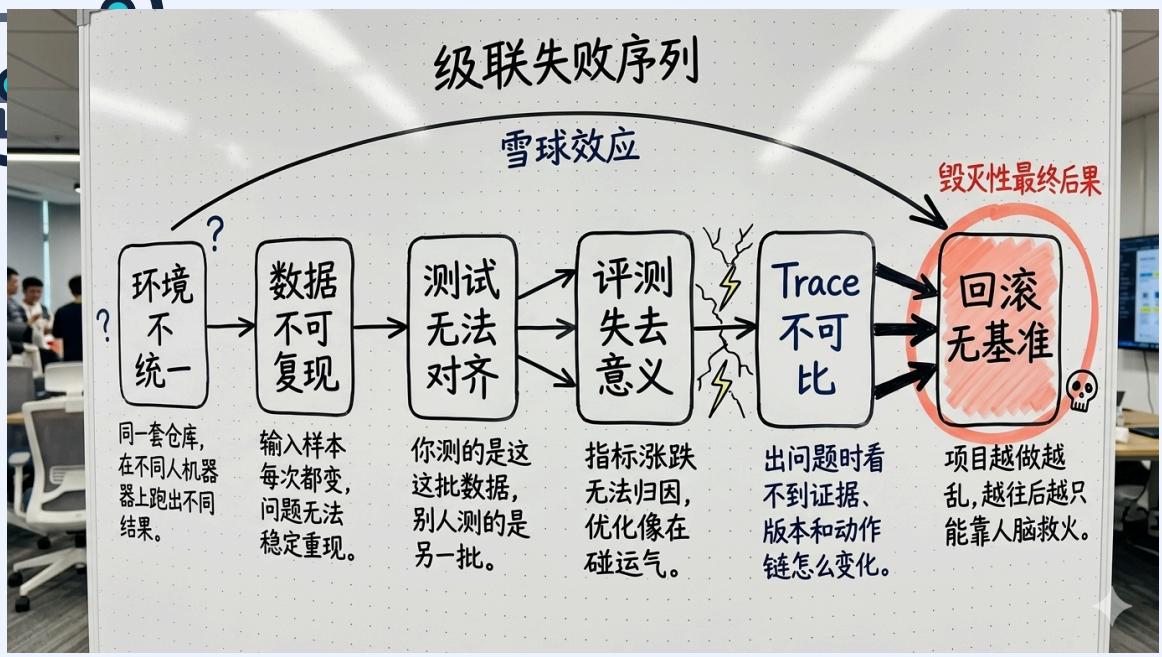


Figure 2

> ! 先记住这句判断
>
> 工程基线不是目录树，而是边界条件。它先把环境、输入、测试、观测和回滚基准写死，后面功能才有稳定土壤可长。

三类边界：工程基线真正要写的东西

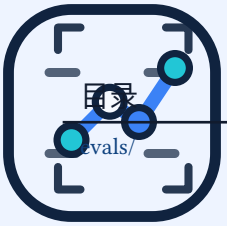
这一段先别急着背 infra、contracts、runbooks 这些目录名。你更应该先抓住三类边界，因为真正决定系统能不能开工的，不是名词齐不齐，而是边界有没有先写清。

Figure 3

项目骨架不是目录树，而是生产问题映射

OmniSupport Copilot 不是单纯的问答机器人，它要同时支持文档问答、证据引用、工单查询 / 创建 / 更新、指标查询、人工介入、审计追踪、回归评测和版本回滚。看项目骨架时，真正要问的不是“目录齐不齐”，而是“它到底在解决什么生产问题”。

目录	在课程里的职责	它解决的生产问题
infra/	环境统一与基础依赖	为什么别人机器上跑不起来?
contracts/	输入输出与动作约束	哪些规则只是口头说的，没有被系统执行?
pipelines/	数据资产加工与编排	输入输出怎么可回放、可重试、可追溯?
services/	RAG 与工具服务化	能力如何暴露成稳定接口，而不是散落脚本?
observability/	tracing / Phoenix / dashboards	出问题时如何定位证据、版本、动作链?



目录	在课程里的职责	它解决的生产问题
evals/	评测集与回归机制	这次优化到底有没有真的变好?
runbooks/	运维与故障处理手册	事故发生后，谁按什么步骤恢复?
blueprints/	落地蓝图与共识文档	团队、讲师、后续代理实现凭什么对齐世界观?

> **Note**
>
> 目录名称不是重点，背后的生产问题才是重点。真正的项目骨架，应该让你一眼看出“这个仓库在为哪类上线风险预留位置”。

七层架构：从输入到负责

七层架构不是为了显得复杂，而是为了提醒你：企业 AI 不可能只盯着检索和生成。越往前，越在保事实；越往后，越在保负责。

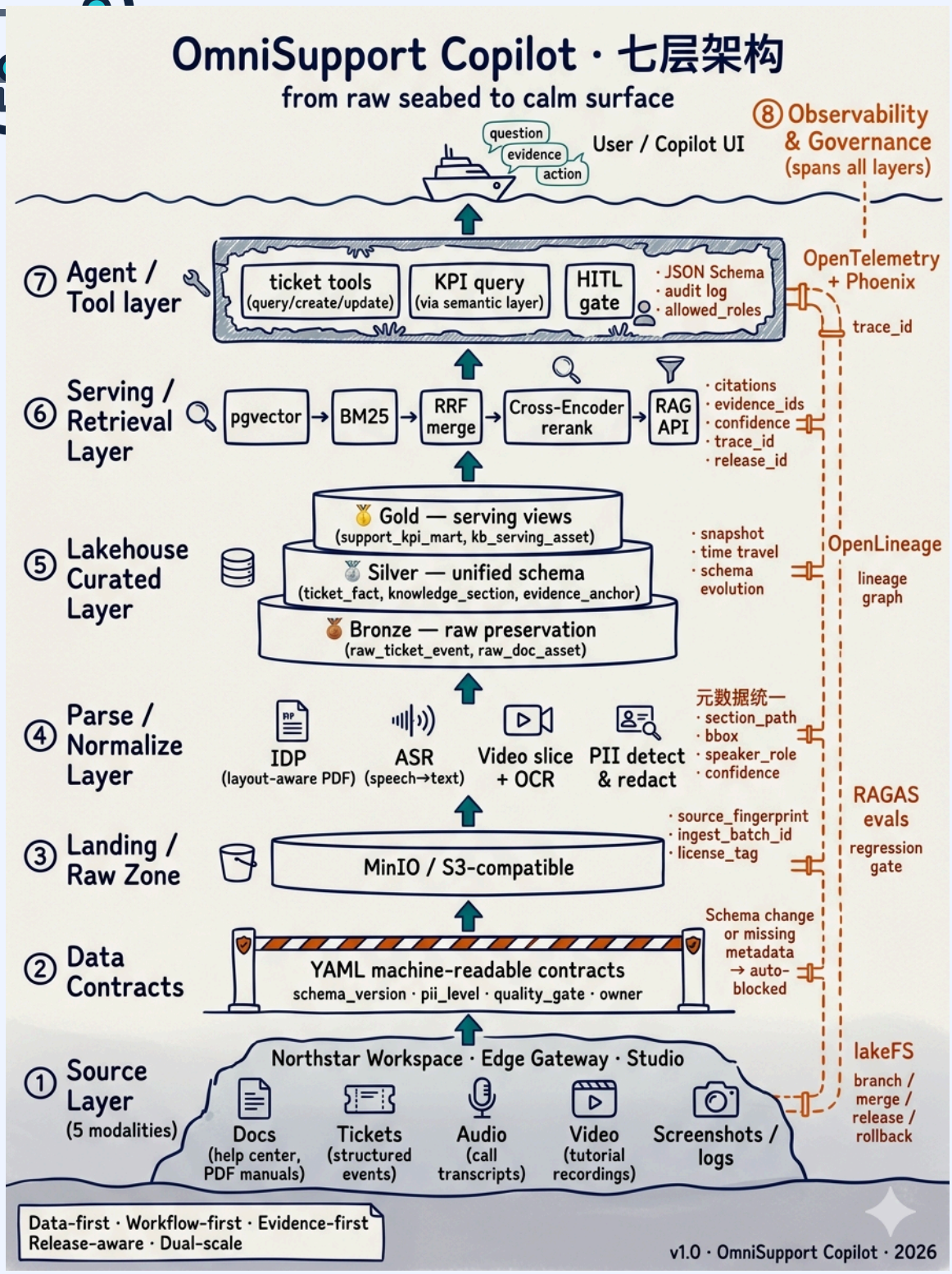
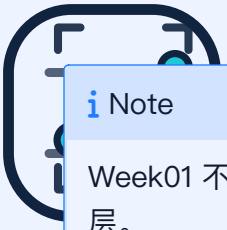

OmniSupport Copilot · 七层架构
from raw seabed to calm surface
⑧ Observability & Governance (spans all layers)
question
evidence
action
User / Copilot UI
⑦ Agent / Tool layer
ticket tools (query/create/update)
KPI query (via semantic layer)
HITL gate
· JSON Schema
· audit log
· allowed_roles
OpenTelemetry + Phoenix
trace_id
⑥ Serving / Retrieval Layer
pgvector
BM25
RRF merge
Cross-Encoder rerank
RAG API
· citations
· evidence_ids
· confidence
· trace_id
· release_id
⑤ Lakehouse Curated Layer
Gold — serving views (support_kpi_mart, kb_serving_asset)
Silver — unified schema (ticket_fact, knowledge_section, evidence_anchor)
Bronze — raw preservation (raw_ticket_event, raw_doc_asset)
· snapshot
· time travel
· schema evolution
OpenLineage
lineage graph
④ Parse / Normalize Layer
IDP (layout-aware PDF)
ASR (speech→text)
Video slice + OCR
PII detect & redact
元数据统一
· section_path
· bbox
· speaker_role
· confidence
RAGAS evals
regression gate
③ Landing / Raw Zone
MinIO / S3-compatible
· source_fingerprint
· ingest_batch_id
· license_tag
② Data Contracts
YAML machine-readable contracts
schema_version · pii_level · quality_gate · owner
Schema change or missing metadata → auto-blocked
① Source Layer (5 modalities)
Northstar Workspace · Edge Gateway · Studio
Docs (help center, PDF manuals)
Tickets (structured events)
Audio (call transcripts)
Video (tutorial recordings)
Screenshots / logs
lakeFS
branch / merge / release / rollback
Data-first · Workflow-first · Evidence-first
Release-aware · Dual-scale
v1.0 · OmniSupport Copilot · 2026

Figure 4



> **Note**
>
> Week01 不要求把七层全部做完，但要求你知道：现在自己站在哪一层，缺的又是哪一层。

六步工程基线演示：把启动手册讲成六个工程承诺

下面这六步，不是为了凑一个启动教程，而是把项目从“看起来像项目”推进到“可协作、可演示、可验证”的工程起点。

> **实操口径来源**
>
> 这一段现在默认对齐 OmniSupport Copilot 的 Week01 启动手册。 后续实验页里的命令顺序和验收口径，也应该以这份 runbook 为准：
>
> - OmniSupport Copilot | runbooks/week01-startup.md

配置隔离

动作

复制团队共享的环境字段模板，并在本地生成可修改配置。

```
cp infra/env/.env.example infra/env/.env.local
```

预期结果

- infra/env/.env.local 已生成
- 团队共享的是字段契约，不是明文机密
- ANTHROPIC_API_KEY 可以暂时留空，Week01 先验证工程基线

它代表的工程承诺

- 机密不写死在代码里
- 本地 / 测试 / 线上配置分离
- .env.example 是协作边界的一部分

环境同构

动作

用同一套 Compose 配置启动服务、网络、卷与依赖。

```
docker compose --env-file infra/env/.env.local \
  -f infra/docker-compose.yml up -d --build
```

预期结果

- 团队成员不再各自拼装本地依赖
- 开发 / 测试 / 演示环境尽量同构
- postgres / minio / rag_api / tool_api / dagster / otel_collector / phoenix 正常启动
- minio_init 作为一次性初始化任务成功退出

它代表的工程承诺

- 环境差异被收敛，而不是留给个人经验兜底
- “别人电脑上跑不起来”不再成为第一周常态
- Week01 先保证学员只装 Docker 就能启动，而不是先被本地依赖卡住

能力可验

动作

先验证关键能力是否可达，而不是只看进程有没有启动。

```
curl http://localhost:8000/health   # RAG API
curl http://localhost:8001/health   # Tool API
```

在浏览器中打开：

- http://localhost:3000 | Dagster UI
- http://localhost:9001 | MinIO Console
- http://localhost:6006 | Phoenix

预期结果

- RAG API 和 Tool API 能返回健康状态
- MinIO Console 可登录并看到初始化后的 bucket
- Dagster、MinIO、Phoenix 都已经进入链路视野

它代表的工程承诺

- 健康检查关心的是关键能力可达，而不是“进程还活着”
- 可观测性不是后面再补，而是第一周就要入场

输入可复现

动作

生成一批稳定的种子工单数据，给后续测试、演示和回归共用。

```
docker compose --profile tools --env-file infra/env/.env.local \
  -f infra/docker-compose.yml run --rm devbox \
  python data/synthetic_generators/ticket_simulator.py \
  --count 500 \
  --output data/canonization/tickets/tickets-seed-001.jsonl
```

预期结果

- 获得可重复使用的 tickets-seed-001.jsonl
- 不需要一开始就接入真实工单数据，也能把底座跑通
- 学员不需要先在宿主机安装 Python 依赖

它代表的工程承诺

- 后续评测、回归、演示有统一输入基线
- 权限和合规门槛可以被逐步引入，而不是 Day 1 就卡死
- Student Core Pack 先把“本地可跑”跑通，再逐步向 Instructor Scale Pack 过渡

边界可检查

动作

先用 seed loader dry-run 和 contract tests 验证输入输出和系统边界已经被写下来。

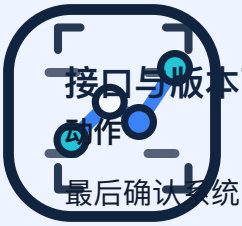
```
docker compose --profile tools --env-file infra/env/.env.local \
  -f infra/docker-compose.yml run --rm devbox \
  python -m pipelines.ingestion.seed_loader \
  --manifest-dir data/seed_manifests

docker compose --profile tools --env-file infra/env/.env.local \
  -f infra/docker-compose.yml run --rm devbox \
  pytest tests/contract/ -v
```

预期结果

- seed loader 能看到 manifest 被接受而不是被拒绝
- 关键契约被自动化检查
- 团队不再只靠口头共识维持系统边界

它代表的工程承诺

- AI 项目不能把测试留到最后
- 比“回答顺不顺”更早的准入门槛，是边界有没有被写成可验证规则
- “contract 在仓库里存在”不算完成，能被 dry-run 和测试验证才算完成



接口与版本可追踪

动作

最后确认系统已经能返回最小响应，并且带出可追踪的版本证据。

```
curl -s -X POST http://localhost:8000/api/v1/query \
  -H "Content-Type: application/json" \
  -d '{"query": "如何配置 Northstar Workspace SSO？"}'

curl -s http://localhost:8000/api/v1/admin/release
```

预期结果

- `/query` 能返回最小可用响应
- 响应中带有 `release_id` 和 `trace_id`
- `release manifest` 能说明当前系统到底跑的是哪一个版本

它代表的工程承诺

- Week01 不只是“服务跑起来”，而是“结果能留痕、版本能追踪”
- 后面做评测、Tracing、回放和发布时，已经有统一版本锚点

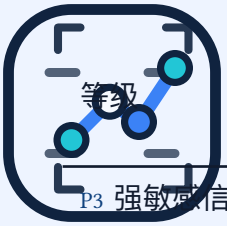
> ! 把六步浓缩成一句话
>
> Week01 的基线演示，真正演示的是六个承诺：配置隔离、环境同构、能力可验、输入可复现、边界可检查、接口与版本可追踪。

风险边界：不是法务附录，而是系统行为开关

AI 项目最危险的，不是它不会做，而是它看起来会做，于是团队忘了先定义“哪些不能做”。风险边界不只是写给法务，它本质上是在定义系统行为开关。

PII 分级

等级	示例	处理原则	是否可进入模型上下文
P0 公共信息	产品手册、公开 FAQ	可直接使用	可以
P1 低敏业务信息	非实名工单摘要、匿名统计	脱敏后使用	视情况
P2 高敏客户信息	邮箱、手机号、设备序列号	工具侧受控读取	原则上不直接进入



等级	示例	处理原则	是否可进入模型上下文
P3 强敏感信息	身份证、支付数据、密钥	禁止进入模型上下文	不可以

PII 分级写进系统之后，后面做解析、契约和服务化时才知道哪些字段能进模型，哪些字段只能在工具侧受控查询。

动作边界

动作类型	示例	默认策略	系统含义
可执行	查询知识、查询工单状态、生成回复草稿	允许	默认可自动完成
受限可执行	创建工单、更新低风险字段	条件允许	需满足权限、证据和状态约束
高风险动作	回滚生产配置、关闭高优先级事故单	必须 HITL	进入人工审批，不自动放行
禁止动作	越权查看敏感订阅、跳过证据直接下结论	明确禁止	系统应直接拒绝或降级

动作边界的核心不是“列个表”，而是让系统在执行前先判断：我有没有权利做这件事。

HITL 触发

触发条件	例子	系统动作	记录要求
高风险操作	请求直接回滚网关配置	转人工审批	记录审批人、证据、时间
证据不足	检索结果无法形成可信建议	输出保守建议 + 请求人工确认	记录缺失证据和 trace_id
状态冲突	文档建议与工单系统状态冲突	停止执行 + 升级处理	记录冲突来源和上下文
权限不足	请求动作超出订阅或角色范围	拒绝执行 + 返回说明	记录权限判断依据

HITL 不是系统失败，而是系统知道什么时候不该自己做决定。

⚠ 把 E1042 放回这张表里看

如果系统只是回答文档步骤，它可以生成建议草稿；但一旦涉及“是否允许自动回滚”，就必须查询权限、判断风险等级，并决定是否进入 HITL。这就是风险边界为什么必须在第一周就写进系统。

Week01 要交的两份工件

Week01 不是只要求“听懂了”。它要求你真正留下两份能被团队、讲师和后续实现共同复用的工件。

交付物 1 | 业务验收口径 & 风险边界清单

为什么现在就交

- 把业务目标、验收口径和系统禁区写成共同语言
- 让后续数据、契约、API、HITL 都能对齐到同一张边界表

最低必填项

- 业务目标、目标用户、高价值场景、禁止场景
- PII 分级、动作分级、HITL 节点
- 失败与降级策略、验收口径

老师会重点看

- 业务口径是否可验收，而不是只剩抽象目标
- 风险边界是否真能落进系统，而不是停留在文档口号

后续怎么复用

- Week02 的数据契约
- Week03 / Week04 的服务契约与接口行为
- Week08 / Week14 的治理、发布和回滚基准

```
## 业务验收口径 & 风险边界清单 v1
- 业务目标:
- 目标用户:
- 高价值场景:
- 禁止场景:
- PII 分级:
- 可执行动作:
- 不可执行动作:
- HITL 节点:
```

```
- 失败与降级策略:
- 验收口径:
```

交付物 2｜AI 系统落地蓝图

为什么现在就交

- 给后续 14 周所有实现提供共同世界观
- 让团队知道当前站在哪一层，后面准备补到哪一层

最低必填项

- 一句话定义、业务世界观、核心对象
- 七层架构、技术选型理由、首周运行基线
- 后续 14 周演进路线

首周运行基线至少应写到

- Docker-only 启动路径，而不是宿主机手工装依赖
- 健康检查、seed loader dry-run、contract tests
- RAG 冒烟查询、release manifest、版本锚点

老师会重点看

- 蓝图是否围绕生产问题展开，而不是罗列技术名词
- 架构层次、运行基线和后续路线是否能闭环

后续怎么复用

- Week03 的架构与路线判断
- Week07 的解析与证据保留
- Week14 的 release manifest 与治理闭环

```
## AI 系统落地蓝图 v1
- 一句话定义:
- 业务世界观:
- 核心对象:
- 七层架构:
- 技术选型理由:
- 首周运行基线:
- 后续 14 周演进路线:
```

! Week01 的真正收口

Week01 结束时，如果没有这两份工件，后面每一周都缺共同基线。功能可以先长出来，但团队很难围绕同一套边界协作、验收和追责。

本讲小结与 Week02 过桥

- 工程基线不是目录树，而是边界条件。
- 风险边界不是法务附录，而是系统行为开关。
- 落地蓝图不是汇报材料，而是后续实现与验收的共同基准。

Week01 解决的是“这套系统有没有资格开工”；Week02 解决的是“它接进来的数据，配不配进入系统”。

所以进入下一周时，你不再是“先接一点数据试试看”，而是要先回答：哪些输入资产能进系统、哪些规则要写成契约、哪些采集动作必须带着边界进入。