场景扩展评估轴：核心对象与人类操作流程（重写版）

先回应你的三条：

1. **数学抽象那篇作废。** 它把"轴类型 / 场景实例 / 依赖 / 结果"翻译成 S、A、K、P、T，翻译完信息量为零，反而把你关注的字段全部藏进了符号里。不再维护。
2. **上一版设计稿的问题**：把你关注的东西拆散塞进了通用词里——枚举值藏在 `output_schema`，发动时间藏在 `inputs` 和 `after` 两处，获取信息藏在 `input_schema` + binding，场景描述藏在 `config_schema`。名词是对的，但没有一个地方把它们并排列出来。
3. 下面按"人什么时候做什么"和"核心对象有哪几个、每个对象里放什么"重新写。范围不变：只在 `llm_probe` 内，复制轴1/轴2作为试验轴，不动生产链路。

---

1. 一句话模型

> **开发者写"轴类型"（一个 adapter 文件），场景配置者在资料页把轴类型实例化成"场景轴"并写特化描述和依赖，运行器在样本到位、依赖满足时发动每个轴，每轴独立落一条带枚举结论的结果。**

核心对象只有 **4 个**，其他都是附属：

#	对象	谁创建	什么时候创建	存在哪
①	**轴类型 AxisType**	开发者	写代码时，一个类型写一次	`llm_probe/eval_axes/adapters/*.py`
②	**场景轴 ScenarioAxis**	场景配置者	在资料页给某场景添加轴时	`data/llm_probe/eval_axes.json`
③	**样本 Sample**	评测执行者	发起试验时选定	现有样本存储
④	**轴结果 AxisResult**	运行器	每个样本 × 每个启用轴跑完	试验运行记录

---

2. 你关注的每个东西放在哪

这是本文最重要的一张表。

你关注的	放在哪个对象	谁填	能否被场景特化	说明
轴类型	① AxisType.`type_id`	开发者	否	一个 adapter = 一个 type_id，如 `fulfillment`、`carryability`
Adapter 实现（流程）	① AxisType.`run()`	开发者	否	单次 / 多阶段 / 逐项核实，随类型走
枚举值	① AxisType.`verdict_enum`	开发者	**否**（改枚举值 = 新类型）	adapter 代码按这套值产出，所以必须在类型层固定
枚举值描述（默认）	① AxisType.`verdict_enum[].description`	开发者	—	该值的通用判定含义
枚举值描述（特化）	② ScenarioAxis.`verdict_notes`	场景配置者	**是**	只能补充/覆盖描述文字，不能增删值
每轴获取什么信息	① AxisType.`needs` 声明 + ② ScenarioAxis.`inputs` 绑定	开发者声明 / 配置者绑定	绑定可变，需求不可变	类型说"我要 request、answer、上游 X 的 output"；实例说"从哪拿"
场景描述（特化）	② ScenarioAxis.`description`	场景配置者	是	注入 adapter prompt 的场景标准，支持 `{material://...}`
依赖 depend_on	② ScenarioAxis.`depend_on`	场景配置者	是	列出依赖哪些轴、要求它们到

什么状态、要不要读它们的 output ｜

｜ **发动时间** ｜ 由 ② `depend_on` + `trigger_when` **推导**，运行器执行 ｜ 场景配置者 ｜ 是 ｜ 不单独存"时间"，存的是条件；条件满足即发动 ｜

｜ **是否启用** ｜ ② ScenarioAxis.`enabled` ｜ 场景配置者 ｜ 是 ｜ 新增默认 false ｜

｜ **结论** ｜ ④ AxisResult.`verdict` ∈ 该类型枚举 ｜ 运行器 ｜ — ｜ 外加 `status`（succeeded/blocked/failed 等）区分"判出来了"和"没跑成" ｜

上一版把"依赖"拆成 `inputs.source=axis` 和 `after` 两个概念，太绕。**本版合并为一个 `depend_on`**，一条依赖项同时说清三件事：依赖谁、等到什么状态、读不读它的输出。

---

3. 人类什么时候做什么

```
□□□ ──────────────────────────────────────────────────────────────────────→

[□□□]              [□□□]                    [□□□]                [□□□]
 □□□                □□□□□                    □□□□□               □□□+□□□
 □ AxisType           □□□□□□□□□□             □□□+□□□□          □□□□□
 - type_id             - □□□                      □□□□                - □□□□?
 - □□□+□□□□     - □□□□□                □□□□□              - □□□□□□□□?
 - needs□□□□□     - □□□□□□□□                                    - □□□□□□?
 - □□□□             - □ depend_on
 - run() □□           - □□ inputs
                       - enabled=true
```

3.1 开发者：定义一个轴类型（写一次）

在 `adapters/` 下新建一个文件，填齐下面这个结构。这就是你说的"轴类型 adapter，不同场景允许不同实现"中的"实现"部分。

impl/projects/llm_probe/eval_axes/adapters/fulfillment.py

```
AXIS_TYPE = AxisType(
    type_id="fulfillment",
    title="□□□□",
    summary="□□□□□□□□□□□□□□□□□□□□□□□□□□□",

    # ── □□□□□□□□□□□adapter □□□□□□□□□ ──
    verdict_enum=[
        Verdict("met",           "□□□□□□□□"),
        Verdict("partial",       "□□□□□□□□□□□□□□□□□"),
        Verdict("not_met",       "□□□□□□□□"),
        Verdict("not_evaluable", "□□□□□□□□□□□□□□□□□□□□□"),
    ],

    # ── □□□□□□□□□□□□□□□□□□□□□□□□□□ ──
    needs=[
        Need("request", kind="sample_field", required=True),
        Need("answer",  kind="sample_field", required=True),
        Need("context", kind="sample_field", required=False),
        # □□□□□□□□□□□□
```

```
    ],

    # ── □□□□□□□□□□□□□□□□ ──
    scenario_fields=[
        Field("description", "□□□□□□□□□□□ capability □□□", required=True),
        Field("show_schema",  "□□□□□□", required=False),
    ],

    # ── □□□□□□ verdict □□□□□□□□□──
    output_fields=["expectations", "assessment", "rationale", "citations"],

    # ── □□□□□□□□□□□□□□□□□□ ──
    default_depend_on=[],

    limits=ExecutionLimits(llm_calls=..., tool_calls=..., seconds=...),  # □□□1□□□
    run=run_fulfillment,   # □□□□1□□□□□□□□□ → □□ → □□ → □□
)
```

impl/projects/llm_probe/eval_axes/adapters/carryability.py

```
AXIS_TYPE = AxisType(
    type_id="carryability",
    title="□□□",
    summary="□□1□□□/□□□□□□□□□□□□□□□□□
```

```
    run=run_carryability,  # □□□□2□□□□□□ → □□□□ → □□ → □□□□□
)
```

开发者的边界：只写类型，不写场景。类型里不出现 `policy_search` 这种场景名。

3.2 场景配置者：在资料页添加场景轴

资料页路径：`资料页 → 选场景 → 扩展评估轴（试验）→ 添加轴`。

表单只有 6 块，顺序就是填写顺序：

步骤	表单块	对应字段	来源
1	选类型	`type`	下拉，只列注册表里的 type_id
2	命名	`id`、`title`	自填
3	场景描述	`description` + 该类型的其他 `scenario_fields`	按类型动态出字段，支持资料引用
4	枚举特化	`verdict_notes`	列出该类型全部枚举值，每个值旁边一个可选文本框"本场景补充说明"
5	依赖与信息	`depend_on`、`inputs`	按类型 `needs` 出：sample_field 的自动绑定；axis_output 的下拉选本场景内类型匹配的轴
6	启用	`enabled`	默认 false

存下来是这样（`data/llm_probe/eval_axes.json`，YAML 展示）：

```
scenarios:
  policy_search:
    axes:
      - id: fulfillment
        type: fulfillment
        title: □□□□
        enabled: true
        description: |
          □□□□□□□□□□□□□□□□□□□□□□□□□□
          □□□□□ {material://llm_probe/policy-capability}
        verdict_notes:
          partial: "□□□□□□□□□□□□□□□□□ partial"   # □□□□□□
        inputs:
          request: sample.request
          answer:  sample.answer
          context: sample.context
        depend_on: []                     # □□□□□□

      - id: boundary
        type: carryability
        title: □□□
        enabled: true
        description: |
          □□□□□□□□□□□□□□□□□□□□□□□
          □□□□□ {material://llm_probe/policy-boundary}
        verdict_notes: {}                 # □□□□□□□
```

```
    inputs:
      answer:      sample.answer
      fulfillment: axis.fulfillment.output
    depend_on:
      - axis: fulfillment
        require: succeeded            # □□□□□□
        trigger_when:
          verdict_in: [partial, not_met]   # □□□□ not_applicable
```

配置者的边界：能改描述、依赖、启用；不能改枚举值、不能改流程、不能改上限。想要这些 → 找开发者加类型。

3.3 评测执行者：发起试验

1. 选场景、选冻结样本集（先用真实旧回答，不重调被测服务）。
2. 点"运行扩展轴试验"（独立入口，不经过生产 judge）。
3. 看结果页：每个样本 × 每个轴一行，列 = `status / verdict / rationale / citations / 耗时 / 调用数`。
4. 看对照页：同样本上 旧轴1 vs 新 fulfillment、旧轴2 vs 新 carryability。

---

4. 运行器：发动时机与信息流（机器视角）

不做通用 DAG，只做下面这个固定流程：

```
□□ × □□
  |
  ├─ □□□□□ enabled=true □□□□ depend_on □□□□□□□□□□□□ disabled □□
  |
  ├─ depend_on □□□□ ──► □□□□□□□□□
  |       □□fulfillment
  |       □□inputs □□□□ sample □□
  |       □□AxisResult(fulfillment)
  |
  └─ depend_on □□□□ ──► □□
          □□boundary
          □□□□□ depend_on.axis □□□□
          □□□
            require=succeeded □□□□□/blocked/not_applicable ──► □□ blocked□□□
            trigger_when □□□□□ verdict=met□            ──► □□ not_applicable□□□
            □□□□                                         ──► □□
          □□inputs □□□□ sample □□ + □□ output□□□□□□ output □□□□ trace□
          □□AxisResult(boundary)
```

"发动时间"不是一个存的字段，而是运行器对 `depend_on + trigger_when` 的求值结果，落在 AxisResult 的 `started_at` 和 `trigger_decision` 里，事后可查为什么发/为什么没发。

---

5. 轴结果：每轴一条，不合并

```
run_id: ...
```

```
case_id: ...
scenario_id: policy_search
axis_id: boundary
type_id: carryability

status: succeeded           # disabled|waiting|running|succeeded|not_applicable|blocked|failed|cancelled
verdict: in_scope_missed    # □□ ∈ carryability.verdict_enum□status≠succeeded □□ null
verdict_description: "□□□□□□□□□□□□□□□□□□□□"   # □□□□ + □□ verdict_notes □□□□□□

trigger_decision:           # □□□□□/□□□
  depend_on_satisfied: true
  trigger_when_matched: {verdict_in: partial}

inputs_used:                # □□□□□□
  answer: sample.answer@<hash>
  fulfillment: axis.fulfillment.output@<run_id>

output:                     # □□□□□□□□ output_fields
  placements: [...]
  rationale: ...
  citations: [...]
  unresolved: []

config_fingerprint: ...
implementation_fingerprint: ...
started_at / finished_at / usage: ...
execution_trace_ref: ...
```

`status` 和 `verdict` 分开的原因：`not_evaluable`（判了，判不出）是业务结论进 verdict；`failed`（没跑成）是基础设施状态进 status。旧轴2部分条目失败的语义原样保留在 `output.unresolved`，整轴 status 按旧规则定。

---

6. 目录落地

```
impl/projects/llm_probe/eval_axes/
  types.py            # AxisType / Verdict / Need / Dependency / AxisResult □□□□
  registry.py         # □□□□□{"fulfillment": ..., "carryability": ...}
  store.py            # □□ eval_axes.json□□□□type □□□□□ notes □ key □□□
                      #   depend_on □□□□□ enabled□□□□□axis_output □□□□
  runner.py           # □□□□□trigger □□□□□□□□□
  adapters/
    fulfillment.py    # □□□1□□□□□□□□□
    carryability.py   # □□□2□□□□□□□□□

impl/data/llm_probe/eval_axes.json
```

复用：Agno / LlmClient / material_tools / 结构化校验 / 上下文记录。不新建第二套底座。

---

7. 首版验收

项	判定
类型可见	资料页下拉能看到 2 个类型，各自枚举值和描述与本文一致
特化可见	场景配置的 description 和 verdict_notes 出现在实际 prompt 与结果的 verdict_description 里
依赖生效	fulfillment=met 时 boundary 落 not_applicable；fulfillment failed 时 boundary 落 blocked；均不发 LLM 调用
信息隔离	boundary 只拿到 fulfillment 的 output，拿不到其 trace
复制等价	冻结样本上，新 fulfillment 的 expectations/assessment、新 carryability 的 placements 与旧链路对照，差异逐条有解释
生产隔离	不进试验入口时，生产轴1/轴2零改动

---

8. 和上一版的差异（只列会影响你理解的）

- 删掉数学抽象文档。
- `inputs` + `after` 两个依赖概念 → 合并为 `depend_on`（含 `require`、`trigger_when`、是否读 output）。
- 枚举值和枚举描述从 `output_schema` 里提出来，成为 AxisType 的一级字段 `verdict_enum`；场景层新增 `verdict_notes` 做特化。
- 类型的"需要什么信息"从 `input_schema` 改叫 `needs`，并区分 `sample_field` / `axis_output` 两种来源，`axis_output` 必须声明接受的上游类型。
- 结果里新增 `trigger_decision` 和 `inputs_used`，让"什么时候发的、读了什么"可查。
- 明确三个角色的操作边界：开发者管类型/枚举/流程，配置者管描述/依赖/启用，执行者管样本/试验/对照。