

# 电力语义层建设思路

如果把这件事情做好，我甚至会把原来的“电力业务语义层”再提升一个层级，定义成：

**Power Grid Semantic & Reasoning Layer（电力业务语义与推理层）**

它不是传统数据中台，也不是单纯知识图谱，更不是给 LLM 外面包一层 RAG。

真正要解决的问题是：

**把“Excel 世界里的数据”，转换成“AI 能够理解、计算、关联、验证和推理的电力业务世界”。**

这件事情恰恰非常适合有大规模数据治理背景的人来做。

## 一、先说我的核心判断

传统数据平台解决的是：

**数据在哪里？字段是什么？怎么查？**

BI/指标平台进一步解决：

**指标是多少？**

知识图谱解决：

## 对象之间有什么关系？

LLM/RAG解决：

## 怎么理解文本并生成答案？

而你们真正需要的是：

## 一个 AI 可以执行的“电力业务世界模型”。

例如用户问：

“为什么 XX 区今年复电率下降？”

AI 不能简单做向量搜索。

它实际上需要完成：

理解“复电率”



找到复电率指标定义



找到 XX 区时间范围



获取复电事件



计算/核验复电率



按线路 / 设备 / 故障类型拆分



找到异常贡献最大的对象



关联缺陷、停电、自动化、运维数据



寻找因果/相关证据



调用规则库判断



形成解释



引用数据证据

所以核心不是“让 LLM 读懂 Excel”。

而是：

让 AI 能够在结构化的电力业务世界里面工作。

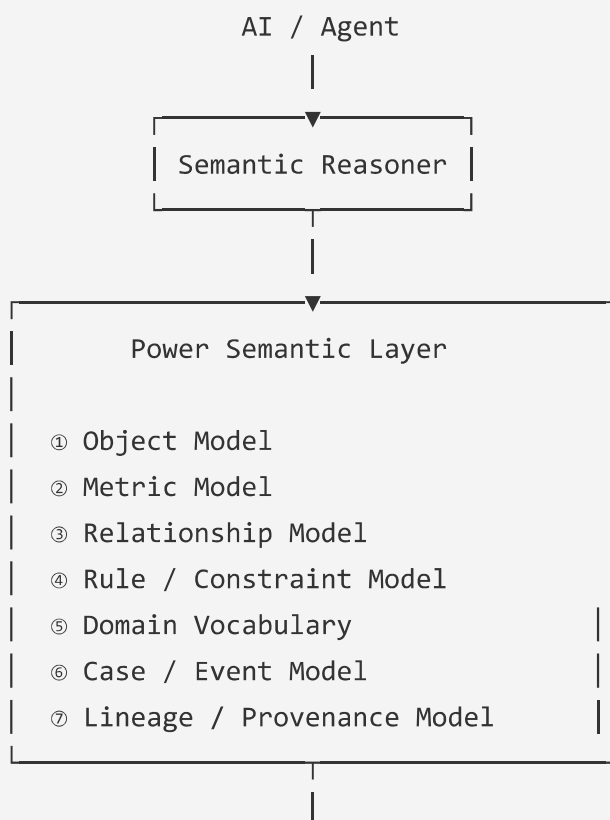
## 二、我会把整个语义层拆成 7 层

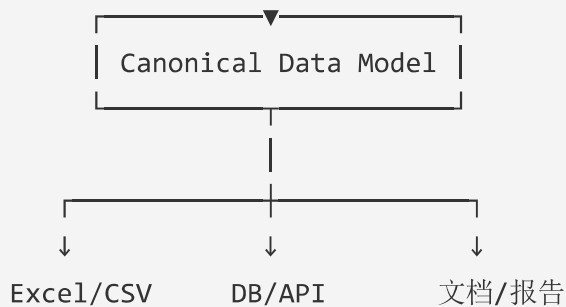
你原来文档里写的：

对象模型、指标模型、关系模型、规则库、专业词典、历史案例库、数据血缘

方向完全正确。

但从工程实现来说，我建议进一步拆成：





这里真正关键的，其实是上面中间那一层。

## 三、第一层：Object Model——先解决“电网里到底有什么东西”

这是整个系统的根。

不要从 Excel 表开始建模型。

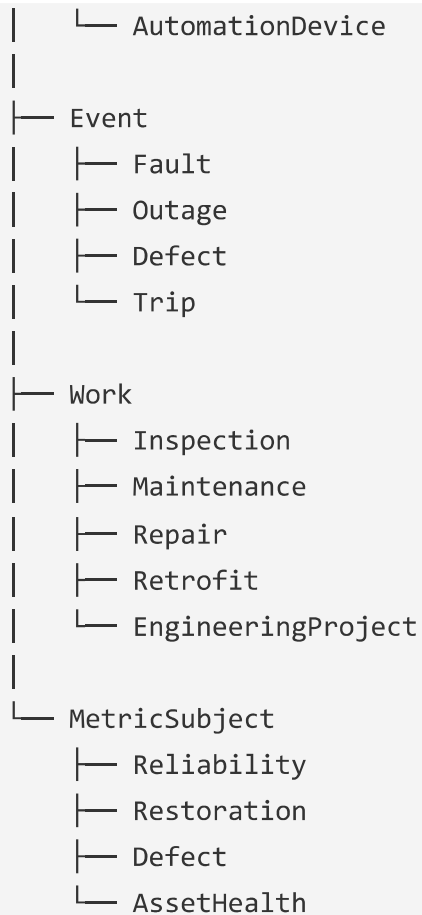
应该从：

### 电力业务对象

开始。

例如：

```
GridObject
├── Organization
│   ├── Company
│   ├── CityBureau
│   ├── District
│   └── Station
├── Asset
│   ├── Transformer
│   ├── Switch
│   ├── Line
│   ├── Feeder
│   └── ProtectionDevice
```



但是这里有一个非常重要的技术问题：

## Object ≠ Table

例如：

```
tbl_equipment
tbl_device_account
tbl_switch
tbl_automation
```

这些可能都是不同表。

但是在业务模型中，它们可能最终指向：

```
Asset
```

所以需要建立：

## Business Canonical Entity

例如：

```
{
  "entity_type": "Switch",
  "entity_id": "SW-123456",
  "name": "XX站10kV出线开关",
  "organization": "XX供电局",
  "voltage_level": "10kV",
  "feeder_id": "F-001",
  "station_id": "S-001",
  "vendor": "...",
  "commission_date": "...",
  "status": "running"
}
```

这样 AI 不需要知道这个数据来自：

设备台账.xlsx  
资产主表.xlsx  
自动化设备.xlsx

AI 只看到：

这是一个 Switch。

## 四、第二层：Relationship Model——真正难的是“关系”

我认为：

**Object Model 只是 20% 的工作，Relationship Model 才是大头。**

因为电网数据的价值，很大程度来自关系。

比如：

开关  
↓ 属于  
线路  
↓ 属于  
变电站  
↓ 服务  
区域  
↓ 发生  
停电事件  
↓ 涉及  
设备  
↓ 导致  
用户影响

同时还有：

设备  
↓  
缺陷  
↓  
检修  
↓  
复电  
↓  
再次缺陷

再例如：

技改项目  
↓ 改造  
设备  
↓ 改变  
设备状态  
↓ 影响  
停电  
↓ 影响  
可靠性指标

所以建议定义一种显式 Relation Model：

```
(entity_A, relation, entity_B)
```

比如：

```
(SW-123, BELONGS_TO, FDR-001)

(FDR-001, CONNECTED_TO, STA-001)

(SW-123, HAS_DEFECT, DEF-20240123)

(DEF-20240123, CAUSED, OUTAGE-001)

(OUTAGE-001, AFFECTED, AREA-001)

(PROJECT-001, MODIFIES, SW-123)
```

## 更重要的一点：

关系最好不是只有三个字段。

应该允许：

```
A --relation--> B

relation:
  start_time
  end_time
  source
  confidence
  valid_status
  provenance
```

因为电网关系是**时态的**。

今天：

开关 A 属于线路 B

明年可能：

开关 A 被改接至线路 C。

所以要做：

### Temporal Semantic Model

这个对 AI 推理非常重要。

## 五、第三层：Metric Model——我认为这是电力 AI 最大的“护城河”之一

这一层经常被传统知识图谱忽略。

比如用户问：

“复电率为什么下降？”

AI 首先必须知道：

**复电率到底怎么算？**

例如可能存在：

```
RestorationRate
=
RestoredCustomersWithinT
/
AffectedCustomers
```

但不同业务部门、不同报表，可能存在不同统计口径。

所以 Metric Model 不只是：

metric\_name

而应该包含：

```
Metric
├─ Definition
├─ Formula
├─ Dimensions
├─ TimeWindow
├─ Grain
├─ Filters
├─ DataSources
├─ Owner
├─ Version
└─ QualityRules
```

例如：

```
metric: restoration_rate
definition: 某统计周期内规定时限内完成复电的用户占比

formula:
  numerator: restored_customers_within_sla
  denominator: affected_customers

dimensions:
  - organization
  - feeder
  - outage_type
  - voltage_level
  - month

time_grain: month

version: 2026.1
```

这样 AI 才能：

**“调用指标”，而不是“猜指标”。**

## 六、这会直接改变 AI 的工作方式

例如：

用户：

“今年复电率下降多少？”

Agent 不应该直接让 LLM 算。

而应该：

```
LLM
↓
Semantic Parser
↓
Metric = restoration_rate
Dimension = year
Organization = current_bureau
↓
Query Planner
↓
Metric Engine
↓
SQL
↓
Result
↓
LLM explanation
```

即：

**LLM 负责理解，Semantic Layer 负责定义，Query Engine 负责计算。**

这是一个非常重要的原则。

## 七、第四层：Rule Model——把专家经验显式化

这一层我觉得特别容易被忽略。

比如：

“同一设备 12 个月内发生 3 次同类缺陷，应重点关注。”

传统做法：

写在专家脑子里。

AI 化之后应该变成机器可执行规则：

```
rule_id: DEFECT_REPEAT_001

condition:
  same_asset: true
  defect_type: same
  count_last_12_months: ">= 3"

severity:
  level: high

action:
  - add_risk_score
  - recommend_inspection
```

于是 AI 就拥有了：

**业务规则执行能力。**

进一步可以做：

## Hard Rule

---

必须满足。

## Soft Rule

---

通常情况下成立。

## Statistical Rule

---

基于历史数据学习。

## Expert Rule

---

专家定义。

最终：

```
Rules
+
Statistics
+
ML
+
LLM
```

形成真正的混合推理。

---

# 八、第五层：专业词典，不只是同义词

---

这个部分如果做得好，会很有价值。

例如：

```
开关
断路器
开关设备
```

CB

断路器设备

可能需要归一到：

CircuitBreaker

但是电力领域更麻烦，因为同一个词有：

**业务含义 + 专业含义 + 系统字段含义**

例如：

主站

可能对应：

- 自动化主站
- 配网主站
- 主站系统
- 某具体系统名称

所以建议词典做成：

Term

└─ Standard Name

└─ Alias

└─ Abbreviation

└─ Domain

└─ Definition

└─ Entity Mapping

└─ Metric Mapping

└─ System Mapping

└─ Confidence

最终可以让 NL2SQL / NL2Graph / Agent 有一层：

**Domain Ontology / Terminology Resolution**

## 九、第六层：历史案例库——不要做成简单RAG

这是我比较想强调的。

很多 AI 项目会说：

“我们把历史报告放进向量数据库。”

这远远不够。

真正有价值的是：

**Case = Situation → Diagnosis → Action → Outcome**

例如：

Case-001

**Situation:**

某馈线连续发生重复跳闸

**Evidence:**

近12个月 5次故障

其中3次发生于雨季

**Diagnosis:**

绝缘问题概率较高

**Action:**

更换绝缘部件 + 防雷改造

**Outcome:**

后续12个月未发生同类事件

**Effect:**

停电次数 -60%

这就是一个**业务案例对象**。

然后 AI 才可以回答：

“有没有类似案例？”

而不是：

“有没有类似文章？”

## 十、因此 Case Model 最好结构化

例如：

```
{
  "case_id": "CASE-001",
  "problem": {
    "type": "repeated_fault",
    "asset_type": "feeder"
  },
  "signals": [
    "high_trip_frequency",
    "rainfall_correlation",
    "aging_insulation"
  ],
  "diagnosis": [
    "insulation_degradation"
  ],
  "actions": [
    "retrofit",
    "lightning_protection"
  ],
  "outcome": {
    "fault_reduction": 0.62
  }
}
```

然后再把文本报告挂在 Case 上。

这比单纯 embedding 文档强很多。

# 十一、第七层：Data Lineage——AI 可信度的生命线

这个非常重要。

因为以后局领导问：

“你为什么说这个设备风险最高？”

AI 必须回答：

我为什么这么判断？

例如：

风险评分 87

↓

设备近24个月缺陷 6次

↓

来源：缺陷系统

↓

设备台账 ID: XXX

↓

最近一次缺陷：2026-07-12

↓

历史同类设备失效概率：...

↓

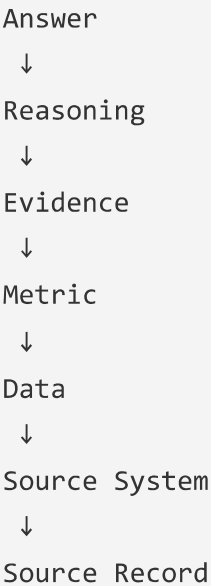
规则：DEFECT\_REPEAT\_001

↓

模型：AssetRiskModel v2.1

所以语义层最终必须具备：

即：



这样 AI 才能进入生产系统。

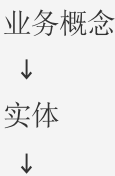
## 十二、我甚至建议不要把 Lineage 只理解成数据血缘

传统血缘：

Excel A → Table B → View C → Dashboard D

AI 时代还要增加：

### Semantic lineage



指标



规则



模型

以及：

## Answer lineage

问题



Agent Plan



Tool Calls



Query



Evidence



Answer

最终一个答案都有：

可追溯推理链。

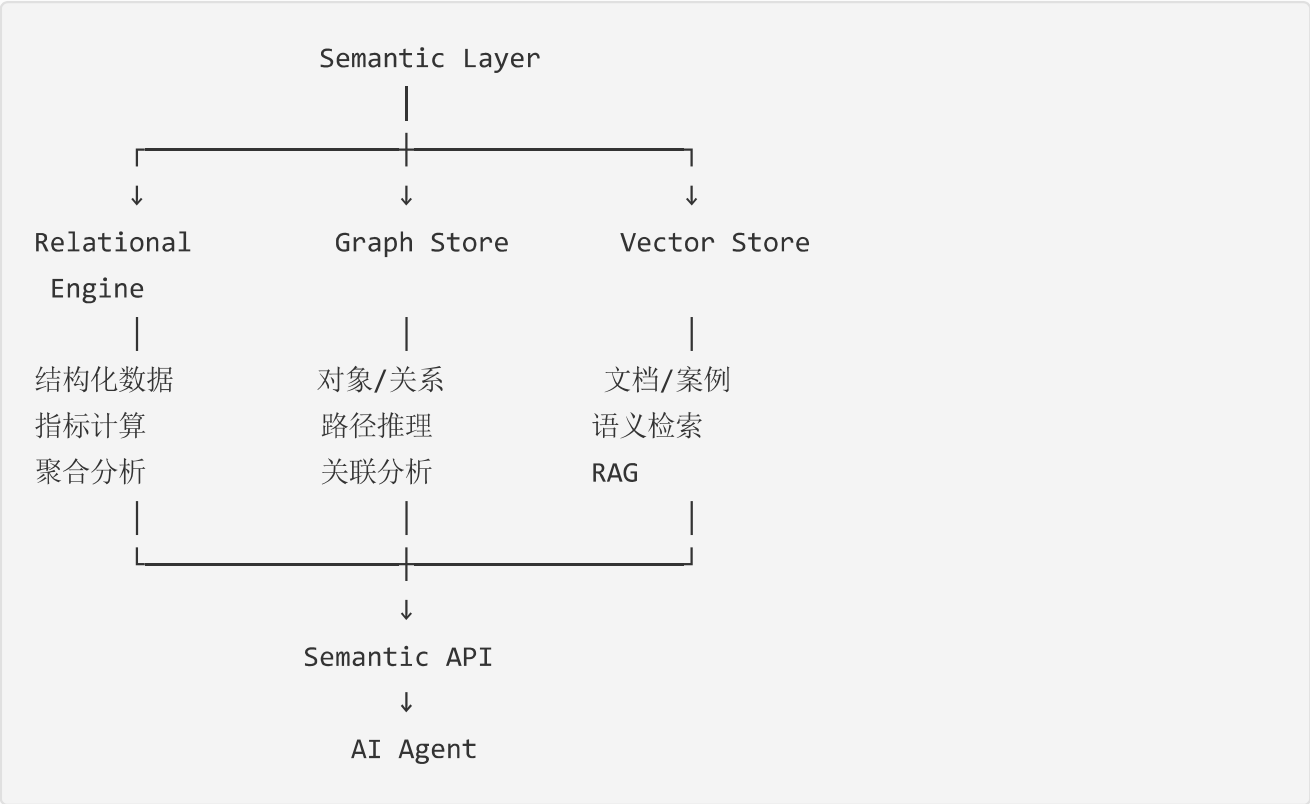
## 十三、技术上，我会采用“四引擎”而不是“一套数据库”

这是我比较强烈的建议。

不要试图：

“把所有东西塞进一个知识图谱。”

更合理的是：



再增加：

Rule Engine  
Feature Store  
Data Quality Engine  
Lineage Service

所以最终实际上是一个：

Polyglot Semantic Architecture

而不是一个“超级数据库”。

# 十四、AI Agent 最重要的是“工具调用”，不是聊天

最终给局里的界面可能就是一个对话框：

“最近三个月哪个供电所的重复缺陷最严重？”

但后台实际可能调用：

```
1. resolve_organization()
2. resolve_metric()
3. query_defects()
4. group_by_asset()
5. detect_repeat_pattern()
6. calculate_risk()
7. retrieve_similar_cases()
8. generate_explanation()
```

所以 Agent 本质上是：

**Orchestrator**

而不是：

Chatbot。

## 十五、我建议你们进一步做一个“Semantic API”

这一层会非常关键。

不要让每个 Agent 自己直接访问底层数据库。

统一提供：

```
/entity/query
/metric/query
/relation/traverse
/rule/evaluate
```

```
/case/search
/evidence/get
/lineage/trace
```

例如：

```
{
  "metric": "restoration_rate",
  "organization": "XX_BUREAU",
  "period": "2026-01/2026-08",
  "dimensions": [
    "feeder",
    "outage_type"
  ]
}
```

返回标准结果。

这样未来无论：

- GPT
- DeepSeek
- Qwen
- 本地模型
- Agent
- BI
- Web 应用

都可以使用同一套语义层。

这会让 Volt Mind 不依赖某一个模型。

---

## 十六、最难的其实不是“建模”，而是实体统一

---

如果让我站在你这个数据平台创始人的角度，我会把最大工程风险排在这里：

### ① Identity Resolution

---

不同系统：

设备A:  
"2023开关柜-001"

设备B:  
"SW001"

设备C:  
"10kV XX线01开关"

到底是不是同一个对象？

这涉及：

Master Data Management + Entity Resolution。

而电网项目里，这往往比 LLM 本身难得多。

## 十七、第二难点是“历史版本”

例如设备：

2022：线路 A  
2024：线路 B  
2026：线路 C

如果没有时间维度：

AI 会把历史世界和现在世界混在一起。

所以必须建立：

**Bitemporal / Temporal Data Model**

至少考虑：

```
valid_time
system_time
```

即：

业务上什么时候生效？

以及：

系统什么时候知道？

这是做严肃电力 AI 必须考虑的问题。

---

## 十八、第三难点是“指标口径治理”

---

比如：

停电次数

不同部门可能：

- 口径 A：统计所有停电
- 口径 B：只统计计划外
- 口径 C：按事件
- 口径 D：按用户
- 口径 E：按线路

如果不做 Metric Governance，AI 可能：

算得很准，但答案是错的。

所以：

Metric Model 的重要性实际上不亚于 Entity Model。

# 十九、第四难点：业务语义层必须“可计算”

这是我最想强调的一点。

很多 Knowledge Graph 项目做到：

设备 → 属于 → 线路

就停了。

但 AI 真正需要：

设备  
↓  
查询历史  
↓  
计算缺陷率  
↓  
比较同类设备  
↓  
应用规则  
↓  
计算风险  
↓  
寻找案例

因此真正的 Semantic Layer 应该是：

**Executable Semantics**

而不是：

**Descriptive Semantics**

区别巨大。

---

## 二十、我甚至会建议最终形成一个 DSL

---

这是更长期的方向。

例如：

```
SELECT feeders
WHERE
    outage_rate > peer_p95
AND repeat_defect_count >= 3
AND automation_coverage < 80%
ORDER BY
    reliability_impact DESC
```

或者更加业务化：

```
FIND feeders
WITH
    high_outage
    AND repeated_defect
    AND low_automation
RANK BY
    improvement_potential
```

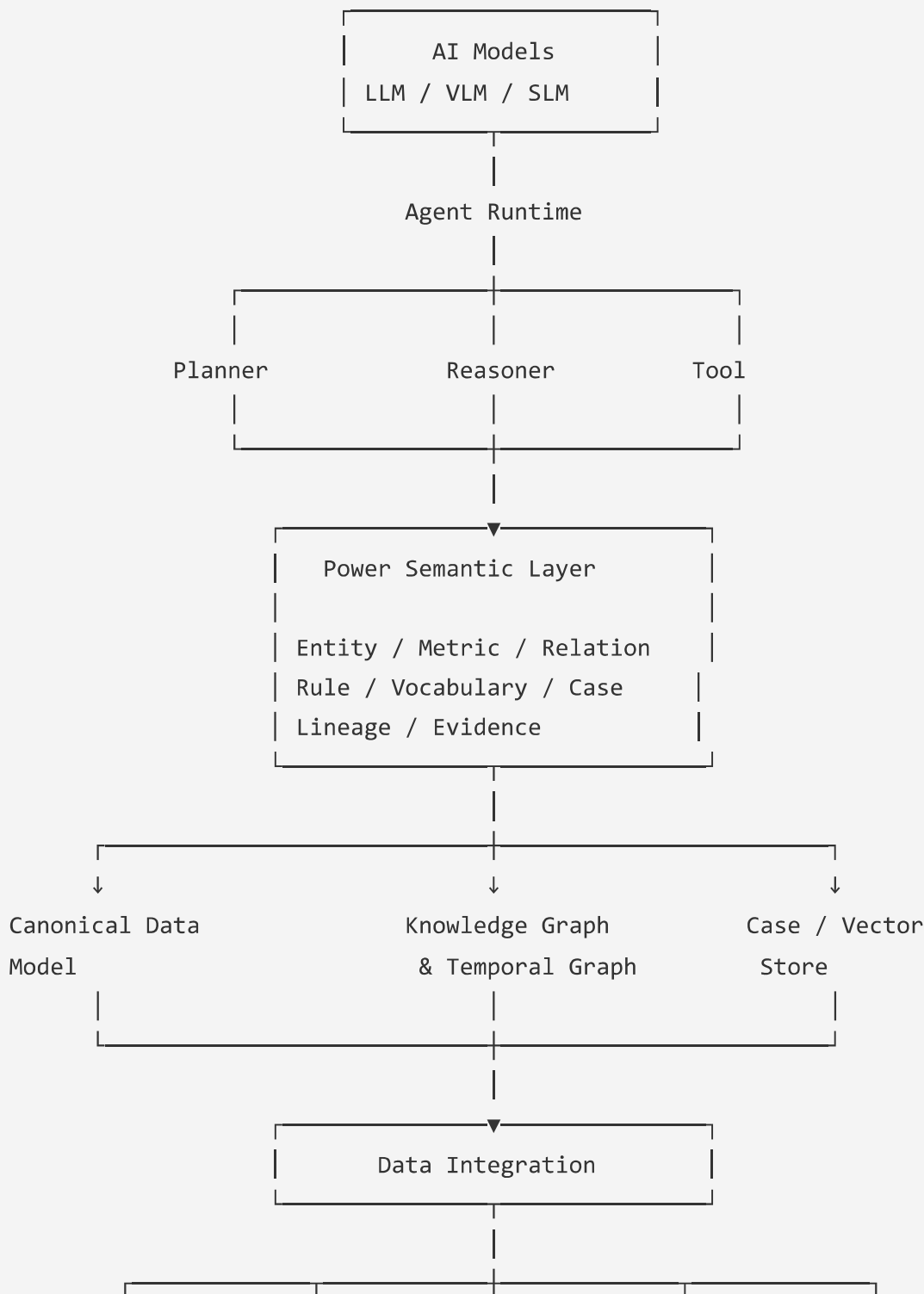
底层编译成：

```
SQL
+
Graph Query
+
Rules
+
ML
```

这实际上就是：

到这一步，Volt Mind 的技术壁垒就开始非常明显了。

## 二十一、最后形成的技术架构，我会这样画



↓  
台账  
运维

↓  
资产  
技改

↓  
缺陷  
自动化

↓  
停电  
防雷

↓  
工程  
项目

而最下面的数据，并不需要全部进入 AI。

真正经过：

清洗 → 标准化 → 主数据 → 实体解析 → 关系识别 → 指标治理 → 语义建模 → 质量校验

之后，才能进入语义层。

## 二十二、如果让我帮 Volt Mind 定义这个产品，我会把它命名成三层

而不是简单叫：

“电力 AI 平台”。

我更倾向于：

### Power Data Fabric

解决：

数据统一

### Power Semantic Layer

解决：

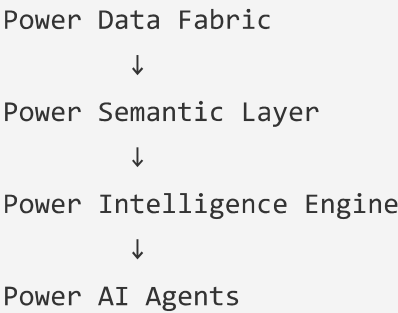
业务统一

## Power Intelligence Engine

解决：

智能统一

最终：



这四层是可以逐步卖给地市局的，而且每一层都有独立价值。

## 最后说一句我认为对你特别重要的

你既然本身熟悉大规模数据治理，那么我会建议不要把这个项目做成一个“AI 项目”。

应该把它定义成：

“电力行业下一代数据基础设施 + AI 原生语义层”

AI 只是最上面的一个应用。

真正的壁垒是：

电网对象模型 + 主数据 + 时态关系 + 指标语义 + 规则 + 案例 + 血缘 + 可执行语义。

一旦这层做扎实，换模型都不是大问题；反过来，如果这层没有做好，即使接上最强的 LLM，也很容易变成一个“会聊天的 BI 系统”。

而且从商业角度看，这个定位对地市局也比“买一个 AI 大模型”更容易讲清楚：**先解决数据资产和生产管理问题，再自然长出 AI。**