

# 大模型时代的知识工程

王昊奋

同济大学 长聘教授

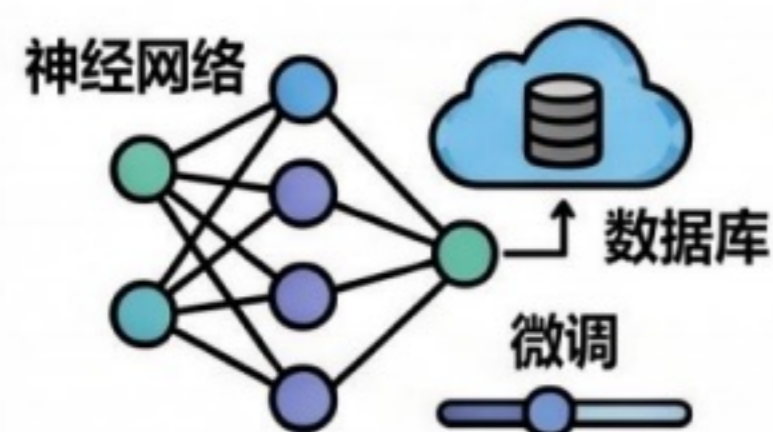
OpenKG 轮值主席

2026.4.17

 公众号 · 开放知识图谱

# 大模型时代不同阶段的知识工程

## PART 1



### 后训练阶段的知识工程

微调、知识注入与模型调整

## PART 2



### 提示工程阶段的知识工程

指令设计、上下文引导与推理控制

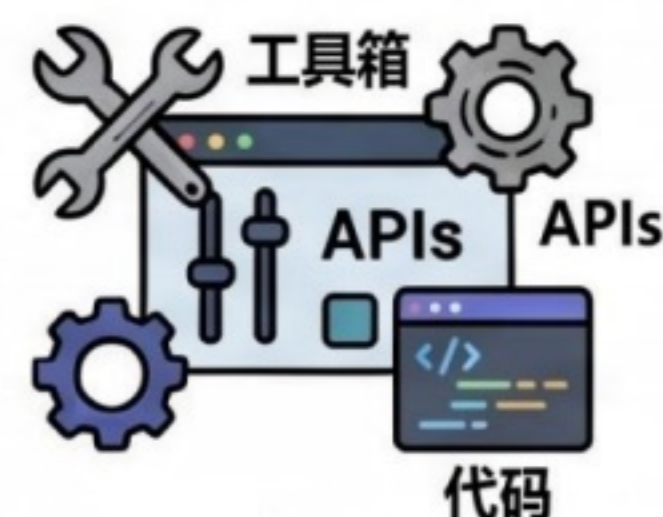
## PART 3



### 上下文工程阶段的知识工程

外部知识库、动态检索与 RAG 集成

## PART 4



### 驾驭工程阶段的知识工程

知识应用、工具集集成与动态执行

# 人工智能知识系统的演进：从规则到智能

## 1.1 专家系统时代 (1970s-1990s)



- 知识 = 规则 (If-Then)
- 知识获取瓶颈：依赖领域专家手动编码
- 代表：MYCIN、DENDRAL
- 核心矛盾：知识获取成本高、维护困难、脆性推理

## 1.2 知识图谱时代 (2000s-2020s)



- 知识 = 三元组 (实体-关系-实体)
- 半自动化构建：信息抽取 + 人工审核
- 代表：Google Knowledge Graph、Wikidata、NELL
- 核心矛盾：Schema 设计僵化、覆盖率与准确率的权衡

## 1.3 大模型时代 (2023-)



- 知识 = 参数化隐式知识 + 外部结构化知识 + 动态上下文
- 自动化构建成为可能：LLM 作为知识工程师
- 核心矛盾：幻觉、时效性、可控性



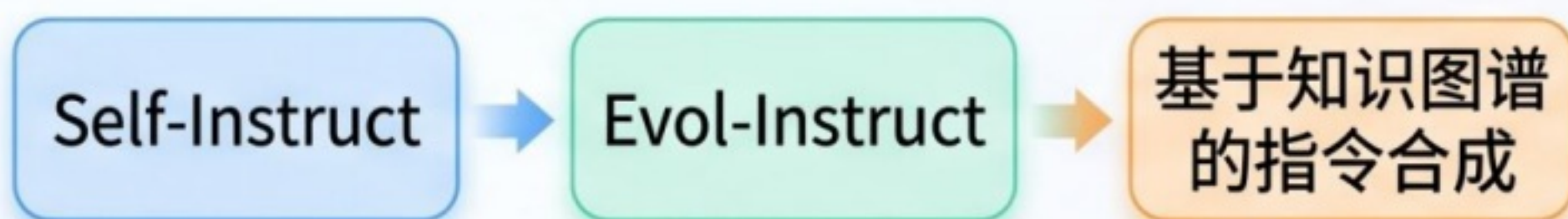
关键洞察：每一次变革都在回答同一个问题——如何让机器更好地利用人类知识？但“知识”本身的定义在持续变化。

# Post-Training 阶段的知识工程

## 知识注入的两种路径

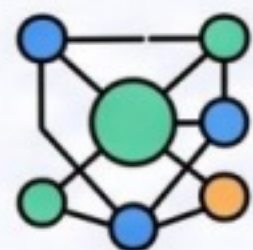
### 指令合成 (Instruction Synthesis)

将领域知识转化为高质量指令-响应对



#### 所需知识形态:

- 领域本体 (定义任务边界与能力范围)



- 技能分类体系 (Skill Taxonomy)

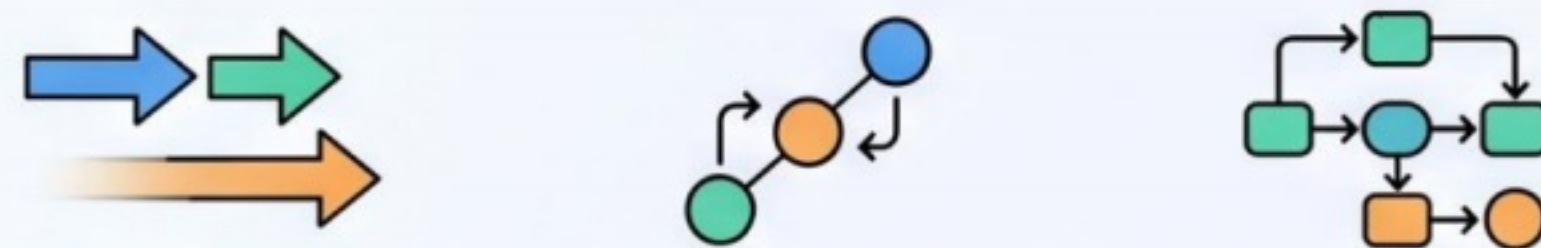


- 质量评估标准 (评判指令质量的元知识)



### 轨迹合成 (Trajectory Synthesis)

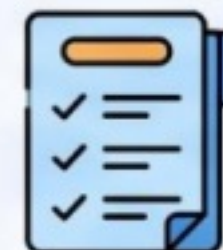
将专家的决策过程编码为 Agent 轨迹数据



- 工具调用序列
- 推理链路
- 多步骤 workflow

#### 所需知识形态:

- 流程知识 (Procedural Knowledge):  
标准操作流程 **SOP**



- 工具语义知识:  
API 语义、调用约束、组合模式

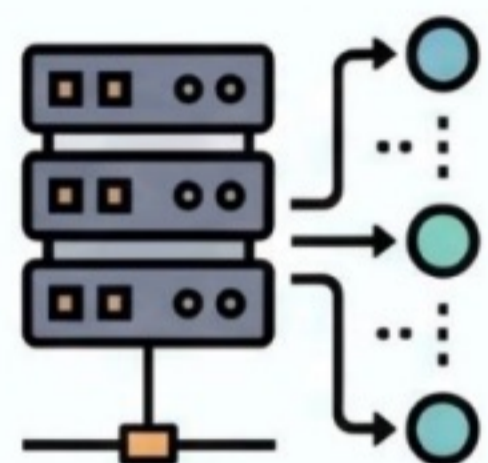


- 策略知识 (Strategic Knowledge):  
何时用什么工具、如何回退



# Post-Training 阶段的知识工程

从静态训练到知识蒸馏



静态训练



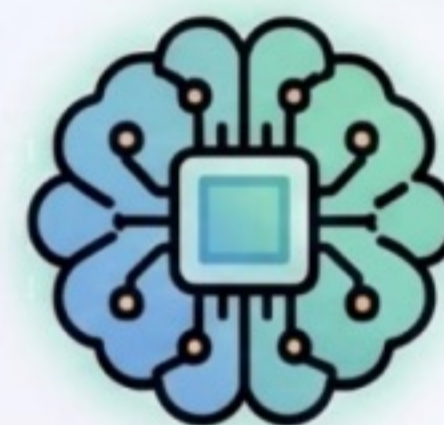
强模型 (LLM)



知识蒸馏

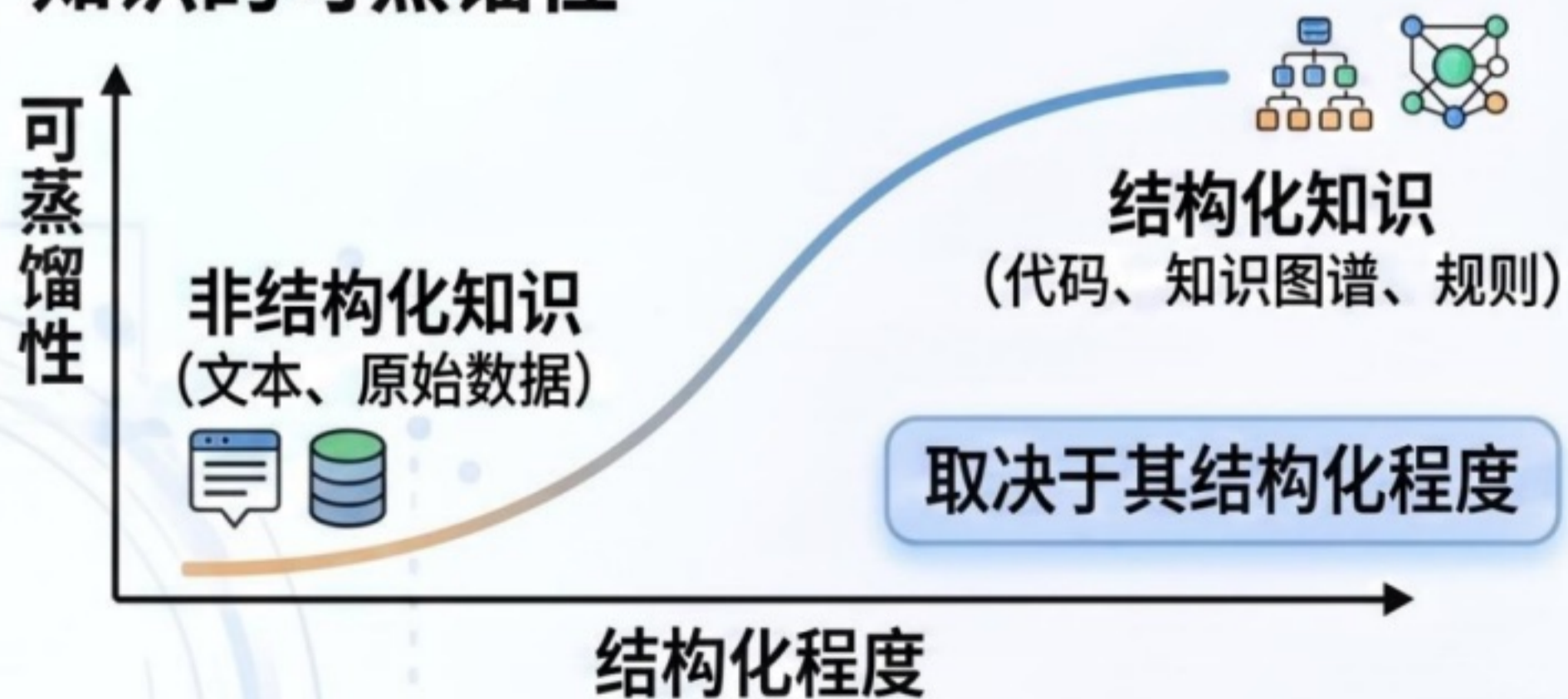


弱模型 (SLM)



知识蒸馏链路

## 知识的可蒸馏性



启示:

越是结构化、可形式化的知识，越容易在 Post-Training 阶段注入



公众号 · 开放知识图谱

# Post-Training 阶段的知识工程

## 所需知识工程实践

知识类型	传统形态	大模型时代形态
领域规则	 专家系统规则库	 指令数据中的隐式约束
实体关系	 知识图谱三元组	 预训练/微调中的参数化表示
流程知识	 工作流引擎	 轨迹数据 (Trajectory)
质量标准	 人工评估指南	 Reward Model 训练数据

# Prompt Engineering 阶段的知识工程

Prompt 的本质是知识的即时注入



## Prompt 的核心元素 & 知识属性



Few-shot examples = 微型知识库



Chain-of-Thought = 推理策略知识的外化



System Prompt = 角色知识 + 行为约束 + 输出规范



## 所需知识形态



**模板知识**: 针对不同任务类型的最优 Prompt 模式



**示例知识**: 精心挑选的 few-shot 样本 (本身就是知识工程)



**元认知知识**: 告诉模型 "你不知道什么"、"何时拒绝"



## Prompt Engineering 的局限性



Prompt 是一次性的、无状态的



不能承载大规模结构化知识



对知识的组织方式高度敏感 (顺序效应、注意力衰减)

转折: 正是 Prompt Engineering 的局限性催生了 Context Engineering.

公众号: 开放知识图谱

# Context Engineering 阶段的知识工程

## 从“写好 Prompt”到“构建正确的上下文”



### Andrej Karpathy 的比喻



LLM 是 CPU



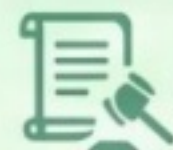
上下文窗口是 RAM



Context Engineering = 内存管理



### 上下文的六大组成要素



**系统指令：**身份、规则、约束



**检索知识：**RAG 拉取的外部信息



**对话历史：**压缩、摘要、  
压缩、摘要、选择性遗忘



**工具定义：**可用能力的声明



**结构化状态：**当前任务进度、中间结果



**示例与模板：**动态选择的参考样本



### 所需知识形态的跃迁



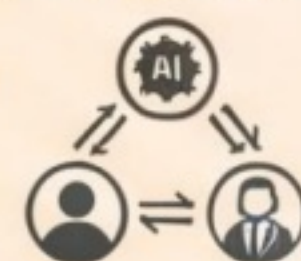
**从静态知识图谱到动态本体**



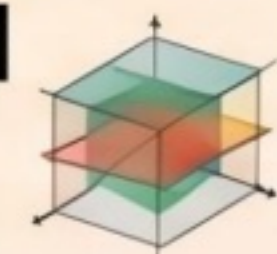
• **Schema 随数据演化**  
新概念自动纳入、旧概念自动退化



• **LLM 驱动的本体维护**  
多 Agent 协作本体工程



• **上下文敏感的本体视图**  
同一知识体系在不同任务  
上下文中呈现不同的切面




**核心转折：知识工程由静态Schema定义转向动态Agent协作本体工程，以适应海量外部上下文。**


# Context Engineering 阶段的知识工程


## 从“写好 Prompt”到“构建正确的上下文”

### RAG 背后的知识工程

**RAG 不只是“检索+生成”，它需要：**


 **索引策略知识：**什么粒度切分、什么信息保留到元数据


 **检索策略知识：**关键词 vs 向量 vs 混合、re-ranking 规则


 **知识冲突处理：**多源信息矛盾时的优先级体系

### GraphRAG 与结构化推理 $\div$


**知识图谱 + 向量检索 + LLM 推理的三层架构**


 **实体消歧知识：**确保实体唯一性

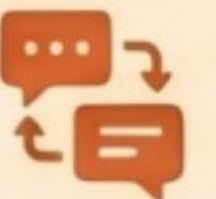
 **关系推理模式：**挖掘实体间间联系

 **社区检测规则：**识别知识社群

### 关键工程挑战

 **上下文窗口**  $\neq$  无限知识，更多 token  $\neq$  更好理解，  
(Chroma 2025 研究证实：过多信息反而降低性能)

 **知识的“渐进式披露”**  
(Progressive Disclosure)，先给地图，按需深入

 **上下文污染** (Context Rot)，长对话中知识退化

公众号 · 开放知识图谱

# Harness Engineering 阶段的知识工程

## 从“给什么信息”到“整个系统如何运转”



### 什么是 Harness Engineering

#### 工程范式系统化:



2026 年初由 Mitchell Hashimoto 命名并系统化



Prompt Eng = What (问什么);  
Context Eng = What (给什么);  
Harness Eng = How (如何运转)



**比喻:** 模型是引擎, 上下文是燃料, Harness 是整辆车 (方向盘、刹车、边界、仪表盘、气囊)

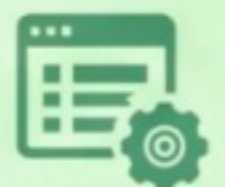


### Harness 的组成要素

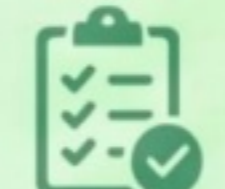
#### 模型周围的一切要素:



**工具与权限** (Tools & Permissions)



**状态管理** (State Management)



**测试、日志、重试** (Testing, Logs, Retry)



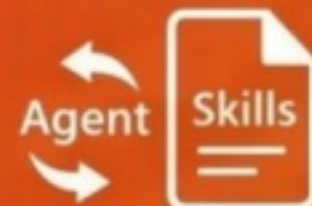
**检查点与护栏** (Checkpoints & Guardrails)



**评审循环** (Review Loops)



### 自我演化的 Harness



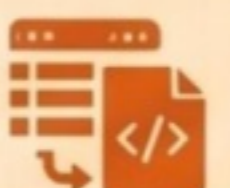
#### 自我优化的系统:



**革命性特征:** Agent 自我改进自己的 Harness



**任务反思执行过程,** 修改技能文件



**提取遥测数据, 自动添加护栏**



**终极形态:** 知识系统不再需要手动维护

公众号: 开放知识图谱

# Harness Engineering 阶段的知识工程



这是知识工程最深刻的变革——知识不再只是‘模型需要知道什么’，而是‘系统应该如何约束、验证、编排模型的行为’。

知识类别	具体内容	载体形式
架构知识	系统的模块边界、依赖关系、不变量	AGENTS.md / CLAUDE.md
验证知识	什么算“正确”、测试用例、质量标准	Linter 规则、CI/CD pipeline
权限知识	Agent能做什么、不能做什么	权限配置、沙箱策略
恢复知识	失败时如何回退、检查点策略	状态机定义、重试逻辑
编排知识	多 Agent协作的角色分工与通信协议	工作流定义、Review Loop
演化知识	从错误中学习并更新 Harness 本身	技能文件自我迭代开放知识图谱

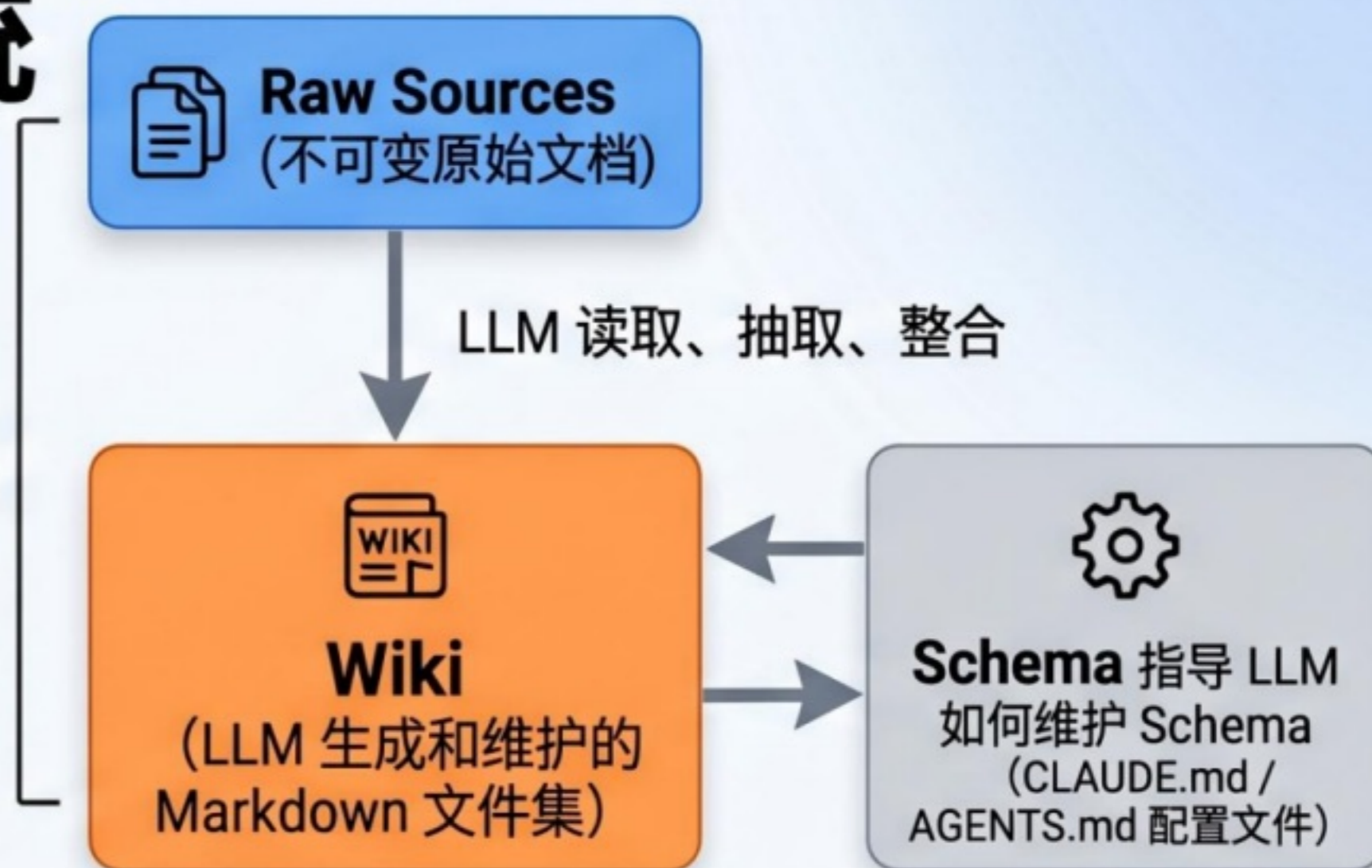
# 新范式：自构建、自演化的知识系统

## Karpathy 的 LLM Wiki 模式 (2026年4月)

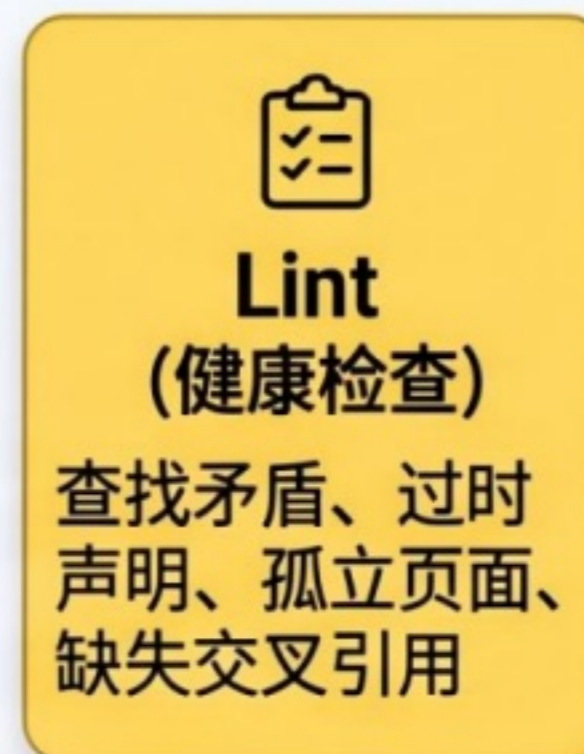
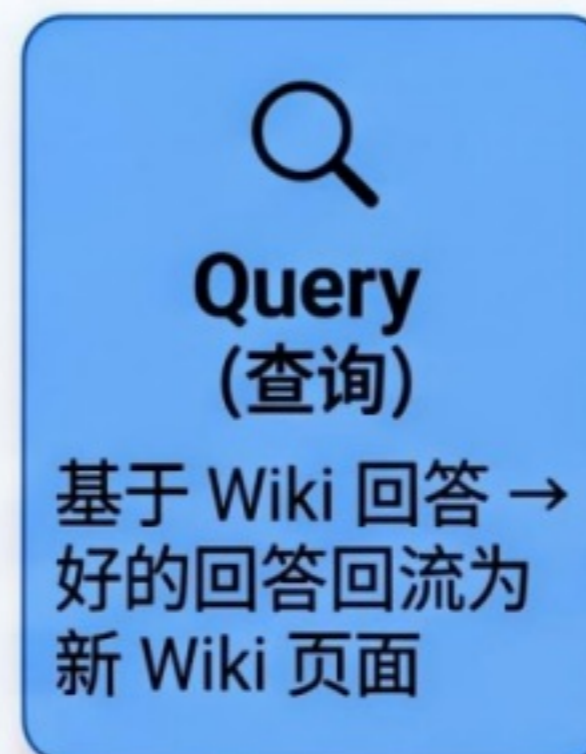
### 核心理念

- 不是 RAG，而是持续编译：LLM 不在查询时从零发现知识，而是增量构建和维护一个持久化的 Wiki
- 知识是累积的产物：交叉引用已建立、矛盾已标记、综合已反映所有已读内容

### 三层架构



### 四个核心操作



### 历史连接

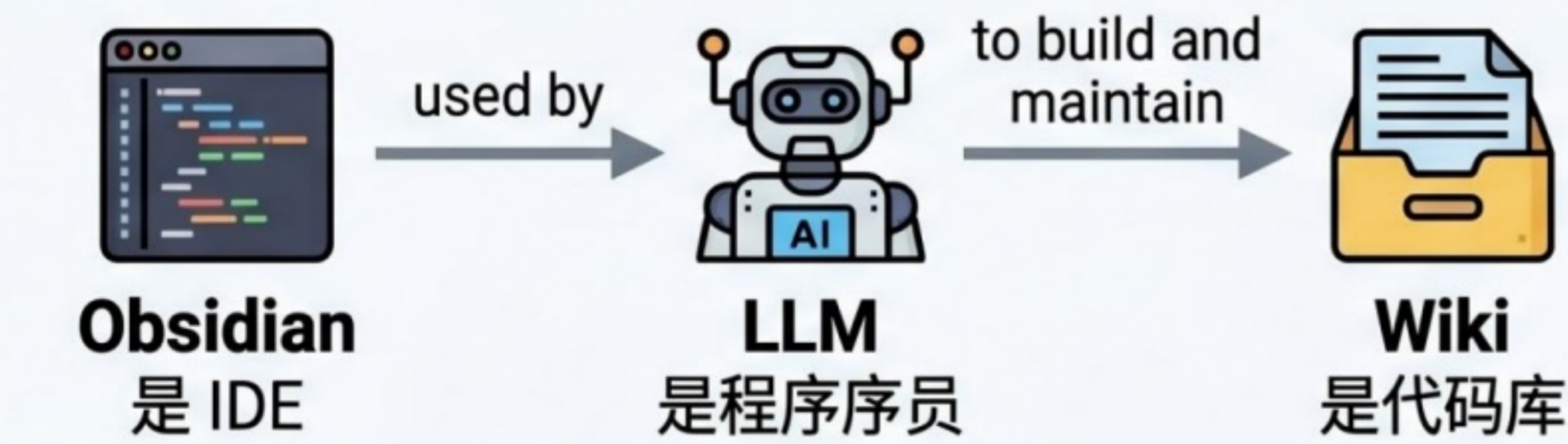
Karpathy 自己将其与 Vannevar Bush 1945 年的 Memex 相对比——一个有关联路径的个人知识存储。Bush 无法解决的部分是“谁来做维护”。LLM 解决了这个问题。

公众号：开放知识图谱

# 新范式：自构建、自演化的知识系统

## Obsidian + Claude Code：知识系统的参考实现

这不只是工具组合，而是一种新的人机共生知识工程范式：



**人类的工作**  
策划来源、引导分析、  
提出好问题、思考意义

**LLM 的工作**  
一切其他事务——摘要、  
交叉引用、归档、簿记

技术栈	组件	角色
Obsidian		知识浏览与可视化（图谱视图、Dataview 查询）
Claude Code Skills		自动化知识维护能力（摄入、更新、健康检查）
Obsidian Web Clipper		原始素材获取
Marp		从 Wiki 内容生成演示文稿
Git		版本控制与协作
qmd / 向量搜索		规模化后的知识检索

# 新范式：自构建、自演化的知识系统

## 从 LLM Wiki 到企业级自演化知识系统



### 个人 Wiki



**来源：**文章、论文、笔记



**维护：**个人 + LLM



**质量保障：**Lint 检查



**本体：**隐式 (Wiki 结构自然涌现)



**知识冲突：**标记矛盾

演化



### 企业知识系统



**来源：**Slack、会议纪要、客户电话、代码库



**维护：**多 Agent + 人工审核环路



**质量保障：**事实验证 Agent + 合规审计



**本体：**Dynamic Ontology (显式但可演化)







**知识冲突：**多源可信度评估 + 时间衰减权重



公众号 · 开放知识图谱

# 知识形态演化全景图

工程阶段 × 知识类型

	 声明性知识 (What is)	 过程性知识 (How to)	 元知识/策略知识 (When/Why)	 演化知识 (Self-improve)
Post-Training	<ul style="list-style-type: none"><li>• 实体关系、事实 (→指令数据)</li></ul>	<ul style="list-style-type: none"><li>• 工具调用轨迹 (→轨迹数据)</li></ul>	<ul style="list-style-type: none"><li>• 奖励模型训练数据 (→对齐数据)</li></ul>	——
Prompt Eng.	<ul style="list-style-type: none"><li>• Few-shot示例</li></ul>	<ul style="list-style-type: none"><li>• CoT推理模板</li></ul>	<ul style="list-style-type: none"><li>• System Prompt约束 (“你不知道就说不知道”)</li></ul>	——
Context Eng.	<ul style="list-style-type: none"><li>• 知识图谱/向量库, Dynamic Ontology</li></ul>	<ul style="list-style-type: none"><li>• RAG检索策略, GraphRAG流程</li></ul>	<ul style="list-style-type: none"><li>• 渐进式披露规则, 上下文窗口管理策略</li></ul>	<ul style="list-style-type: none"><li>• Dynamic Ontology (Schema自动演化)</li></ul>
Harness Eng.	<ul style="list-style-type: none"><li>• AGENTS.md文档, 架构规范</li></ul>	<ul style="list-style-type: none"><li>• CI/CD验证流程, 多Agent编排协议</li></ul>	<ul style="list-style-type: none"><li>• 权限/沙箱策略, 失败恢复策略</li></ul>	<ul style="list-style-type: none"><li>• Skill File自迭代, 从错误中学习更新</li></ul>
LLM Wiki / 自演化系统	<ul style="list-style-type: none"><li>• Wiki页面, 交叉引用网络</li></ul>	<ul style="list-style-type: none"><li>• Ingest/Query/ Lint工作流</li></ul>	<ul style="list-style-type: none"><li>• Schema配置, 质量评估标准</li></ul>	<ul style="list-style-type: none"><li>• LLM自主维护Wiki + 人类引导演化</li></ul>

# 知识形态演化全景图

## 核心趋势

### 载体 (Carrier)



SQL



Markdown

### 从数据库走向文档

SQL > Markdown  
LLM 是文本原生的

### 构建 (Construction)



人工



自动/LLM

### 从人工走向自动

LLM 作为知识工程师

### 维护 (Maintenance)

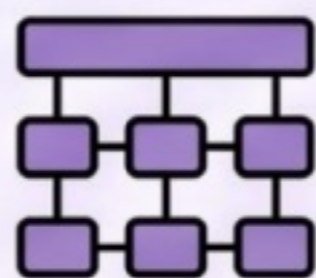


主动

### 从被动走向主动

Lint、健康检查、自动更新

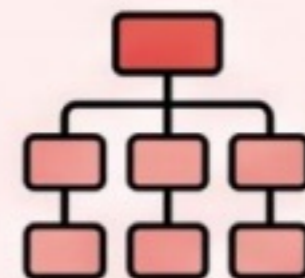
### 形态 (Morphology)



### 从静态走向动态

Dynamic Ontology、自演化 Schema

### 边界 (Boundary)



涌现

### 从预定义走向涌现

Wiki 结构不是预先设计的，而是在使用中自然生长

# 展望与开放问题

## 技术前沿



- **知识编辑 (Model Editing)**：精确修改模型内部单条知识，无需重新训练。
- **多模态知识工程**：图像、视频、代码中的知识纳入统一框架。
- **长上下文 vs 外部知识库**：随着窗口扩展到 100M+ token，RAG 是否还有必要？
- **神经符号混合系统**：知识图谱推理 + LLM 生成深度融合。

## 工程挑战



- **Harness Debt (Harness 技术债)**：Harness 本身成为复杂的维护系统。
- **知识漂移检测**：自演化系统如何不偏离真实？
- **可审计性**：金融/医疗/法律场景下，自动构建知识的合规要求。
- **知识确权**：LLM 合成知识的版权归属。

## 哲学思考



- **人类的角色**：知识系统自我维护、自我演化时，人类的角色是什么？
  - Karpathy：人类负责策划、探索、提出好问题；LLM 负责其余一切。
- **从存储到涌现**：知识不再是被“放进去”的，而是在交互中“长出来”的。
- **知识工程之梦**：1945 Memex → 2001 Semantic Web → 2026 LLM Wiki，80年可行的实现路径。

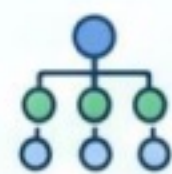


# 结语

## 知识工程的重生



→ 规则，变成了 AGENTS.md 中的约束

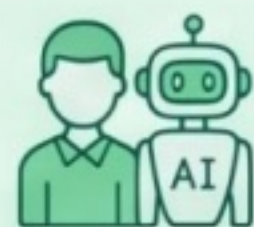


→ 本体，变成了 Dynamic Ontology，在使用中演化



→ 知识图谱，变成了 GraphRAG 的结构化骨架

## 人机协作新范式



从前，知识工程师  
手动编码



现在，LLM 在人类的引导下自  
动构建、自动维护、自动演化



人类的工作从“搬砖”升级为“架构”  
——定义 Schema、设计 Harness、提出正确的问题

# 知识工程已死，知识工程万岁。