# Cross-Disciplinary Knowledge Graph Agent (跨学科知识图谱智能体)

**云计算期末大作业｜智能体云原生开发（命题三）技术文档**

> 本项目面向“知识碎片化、跨学科关联难以发现”的学习痛点：用户输入一个核心概念，系统通过多阶段 LLM Agent 强制跨学科检索与抽取，构建可交互、可解释、可扩展的跨学科知识图谱，并以云原生方式一键部署运行。

# 1. 项目内容介绍

## 1.1 目标与痛点

- **痛点**：同一概念在不同学科（数学/物理/计算机/生物/经济等）中常以不同语言与语境出现，学习者难以看到其内在联系。
- **目标**：输入核心概念（如 *Entropy*、递归），智能体自动在多个学科域强制展开检索、抽取实体与关系，生成结构化知识图谱，并在 Web 端动态可视化与交互探索。

## 1.2 核心功能清单

- **跨学科强制检索**：默认覆盖 Mathematics / Physics / Computer Science / Biology / Economics 等领域，Planner 为每个领域生成多条 query，Retriever 执行多源检索。
- **知识图谱构建**：以标准 JSON（GraphResult）输出，包含 nodes / edges / evidence / meta 等结构，并用 Pydantic 做模式定义与校验。
- **Web 端动态可视化**：React + ECharts 展示跨学科图谱；点击节点/边查看详情与证据；支持筛选域/关系/校验状态；支持导出 JSON。
- **二跳扩展（Expand 2-hop）**：对选中节点进行 2-hop 扩展并合并入图谱，实现“边探索边生长”的动态图谱体验。
- **证据展示与可解释性**：每条边提供 evidence（标题/片段/来源），并展示可信度与校验结论（Pass/Fail/Conflict）。

# 2. 系统架构与数据流（Architecture & Dataflow）

## 2.1 总体架构（前端-后端-Agent-存储-外部知识源）

- **Frontend（React + ECharts）**：用户输入概念、触发生成、交互探索与过滤、导出结果。
- **Backend（FastAPI + Orchestrator）**：承接 API 请求，调度多阶段 Agent，管理异步任务与日志，读写图数据库。
- **Redis**：任务状态与 Job Logs 缓存、历史任务记录；前端轮询 `/api/job/{job_id}` 获取进度与日志。
- **Neo4j**：知识图谱持久化存储；提供按概念查询子图与扩展合并能力。
- **External Data Sources**：Wikipedia / arXiv / Search API 或内置 seed 语料（支持离线 mock）。

## 2.2 核心数据流

1. **用户输入概念**（前端）→ 点击 `Generate`
2. **后端创建任务**：返回 `job_id`；任务进度/日志写入 Redis
3. **Agent 多阶段执行**：Planner → Retriever → Extractor → Validate/Check → Merge/Bridge
4. **持久化**：通过 Neo4jStore 将节点与关系 upsert 入 Neo4j
5. **前端轮询任务结果**：拿到 GraphResult → 渲染主图 + 侧栏详情 + Relationship Graph
6. **交互扩展**：用户点击 `Expand (2-hop)` → 后端调用 expand 接口 → 增量生成/合并 → 更新图谱

# 3. 云原生组件与工程实践（Cloud-Native & Engineering）

## 3.1 Docker 容器化

- 后端： `python:3.11-slim` + Uvicorn 启动 FastAPI 服务
- 前端： `node:20-alpine` + 启动 Vite/前端服务（默认 5173）

## 3.2 Docker Compose 一键编排

Compose 定义 **4 个服务**： `redis` 、 `neo4j` 、 `backend` 、 `frontend` ，并配置端口映射（8000/5173/7474/7687）。
同时利用 `depends_on + healthcheck` 保障依赖服务健康后再启动 backend，提升稳定性。

## 3.3 配置与环境变量（Config as Code）

- 后端 Settings 支持通过环境变量覆盖：Redis 地址、Neo4j 连接、LLM Provider、OpenAI 兼容配置等；提供 `env.example` 作为配置模板。
- Compose 通过环境变量注入关键参数，实现**配置与代码分离**、便于迁移与部署。

## 3.4 健康检查与容错

- 服务健康检查：Redis `redis-cli ping` 、Neo4j cypher shell 查询；后端提供 `/api/health` 用于探针。
- 任务容错：Agent 过程异常捕获写入 Redis logs，并将任务标记失败，避免影响其他请求。

# 4. 智能体策略（Agent Strategy）

## 4.1 多阶段 Agent 工具链

在本项目中，我们设计了清晰拆分的多阶段流程，而非一次性让模型完成所有工作。这种多阶段Agent设计本身就类似隐式的链式思维，引导模型逐步聚焦于子任务：先发散思考查询方向，再根据资料抽取关系，最后检验结果。这种逐步细化的过程减少了单次Prompt的认知负担，降低了模型胡乱编造的概率。

同时，我们在Prompt中严格限定了输出的格式，几乎在每个与LLM交互的阶段都要求返回JSON。例如，Planner和Extractor的系统消息都声明“只返回JSON对象，不要Markdown或解释。这种格式桁架使模型不敢偏离要求，大幅降低了出现自由文本或不符结构内容的风险。
提示词示例：

```
system = (
        "You are a strict JSON repair assistant. "
        "Return ONLY a valid JSON object, no markdown, no commentary."
    )
    user = {
        "task": "Fix the graph JSON to satisfy the schema and constraints.",
        "constraints": [
            "Keep concept/nodes/edges/meta structure.",
            "Each edge must have evidence.title and evidence.snippet (url optional).",
            "Edge.relation must be one of: related_to, used_in, is_a, explains, bridges.",
            "confidence must be within [0,1].",
            "checked must be boolean.",
            "Add missing fields with safe defaults; do not delete nodes/edges unless absolutely
        ],
        "schema_errors": errors[:30],
        "graph": graph_dict,
    }
```

智能体在接收到用户输入的核心概念后，会依次经过 Planner、Retriever、Extractor、Bridge、Checker 等阶段，以多步提示词（Prompt）和工具链协作完成跨学科知识挖掘与图谱构建。整个流程由 Orchestrator 调度，并通过 Redis 实时记录状态日志，最终将结果图谱持久化到 Neo4j，供前端查询展示。

## (1) Planner：跨学科强制规划

负责跨学科查询规划。当用户提供一个核心概念后，Planner 阶段调用 LLM 生成针对此概念的多领域检索查询和候选相关/桥梁概念，以确保强制跨学科扩展，而不是局限于单一领域的同义词。该 Prompt 在收到用户输入后立即触发，用于规划后续检索方向。

- 输入：核心概念、默认领域列表、扩展深度
- 输出：每个领域若干检索 query（3~6 条/域），确保覆盖不同学科语境。
  关键提示词：

```
system = "You are a cross-domain planner. Return ONLY a JSON object."
        payload = {
            "concept": concept,
            "domains": domains,
            "output_format": {"domains": {"Domain": ["query1", "query2", "query3"]}},
            "requirements": [
                "For EACH domain, output 3-6 related concepts/bridge concepts.",
                "Prefer textbook keywords; can include bilingual hint in parentheses.",
                "Do NOT repeat the same query across domains.",
                "Keep queries short (<= 8 words).",
            ],
        }
```

## (2) Retriever：多源检索与证据收集

负责依据 Planner 给出的查询到不同知识源抓取证据片段。每个领域的查询将通过代码调用外部 API（如 Wikipedia、学术论文接口等）获取相关文献或百科片段。这个过程是不直接依赖 LLM 的 Prompt，而是由程序逻辑控制触发，在 Planner 完成后执行。

对于每个领域，系统采用异步爬取或API查询的方式，从 Wikipedia、arXiv论文库以及其它搜索接口检索相关内容片段。每个查询在每个数据源上限定获取少量结果(≤3)，以控制总语料量。

- 对各领域 query 分别检索，收集 passages（默认每源最多 2 条、每领域合并上限等）。 检索的时候会调用外部数据源（Wikipedia、arXIV等）

## (3) Extractor：结构化图谱抽取

负责结构化图谱抽取。在获取各领域的证据文本后，Extractor 阶段调用 LLM 将这些非结构化信息转换为标准化的知识图谱 JSON，包括节点、边、关系解释以及每条边对应的证据引用。Extractor Prompt 在 Retriever 收集到足够证据后触发，将证据作为输入，让 LLM 输出符合预定架构的图谱数据。

- 使用 LLM 从多领域证据中抽取 GraphResult(JSON)：
  - Core 概念必须归入 “Core” 域
  - 至少覆盖 4 个不同领域节点
  - 每条边包含 relation、explanation、evidence（title、snippet、source/domain）
  - snippet 尽量同时提及 source/target 名称以便后续校验通过

关键提示词：

```
payload = {
            "concept": concept,
            "domains": domains,
            "passages_by_domain": passages_by_domain,
            "schema": {
                "concept": "string",
                "nodes": [
                    {"id": "string", "name": "string"......}
                ],
                "edges": [
                    {
                        "id": "string|null",
                        ......
                    }
                ],
                "meta": {"generated_at": "ISO string", "version": "v1", "checker_summary": "obje
            },
            "requirements": [
                "Must include the central concept as a node in domain 'Core'.",
                "Must include at least 4 domains in nodes.",
                "Edges must include evidence.title and evidence.snippet; snippet should mention
                "Use bridges edges to connect distant domains via 1-2 bridge concepts.",
                "Prefer concise, readable node ids. Example: 'mathematics:recurrence_relation'."
            ],
        }
```

## (4) Validate + Checker：双层校验（结构 + 证据一致性）

负责结果校验与解释强化。该阶段在 LLM 给出初步图谱后触发，包含两个层次：模式结构校验和证据一致性校验。Checker 会通过代码逻辑和（可选的）LLM Prompt，对生成的 JSON 进行schema验证和修复，并检查每条边的证据支撑是否充分。例如，如果某关系的证据片段未同时提及关系两端的概念，Checker 会将此边标记为未通过检查（checked=false）或将关系降级为“related_to”以表示弱关联，从而降低大模型幻觉带来的风险。

- **Schema Validate & Light Fix**：用 Pydantic 模型校验 JSON；字段缺失/类型不匹配时尝试自动修复（补字段、补 evidence、生成缺失 id 等），并记录修复统计。
- **Evidence & Consistency Check**：逐边检查：
  - evidence.title/snippet 与 explanation 不能为空
  - snippet 必须提及边两端概念（或关键词），否则判为 unchecked 并下调置信度
  - 输出统计：passed / failed / downgraded / conflicts

## 4.2 Bridge 机制（跨学科桥梁）

- 系统会从多域概念中发现“共享主题/桥梁概念”，并用 `bridges` 关系把不同学科子图连接起来，提升跨域可走通性与解释链构建能力。

## 4.3 去重与冲突处理（Dedup & Conflict）

- **同义节点合并**：标准化节点名（去括号/符号/大小写等）+ 领域键合并同名节点，更新引用。
- **重复边合并**：同 (source,target,relation) 合并，保留高置信版本并融合多证据 snippet。
- **冲突检测**：同一对节点存在多种关系类型则标记 conflict 并降权；前端可过滤高亮 conflict 边。

# 5. 前端可视化与交互设计（UI/UX）

## 5.1 主图谱（Graph Canvas）

- 节点按 Domain 上色，支持点击查看详情；支持拖拽、缩放、搜索定位等。

## 5.2 侧栏详情（Entity/Edge Card）

- 节点：领域、定义、关联关系摘要
- 边：关系类型、解释、证据片段（最多 3 条）、来源链接、置信度、校验状态（Pass/Fail/Conflict）

## 5.3 Relationship Graph（局部关系视图）

- 将当前选中节点/边的一跳关系抽出单独渲染，便于讲清单条关系与证据。

## 5.4 Filters（分析过滤）

- **Domain**：按学科域筛选（含 Core/Bridge）
- **Relation**：按关系类型筛选（related_to/used_in/is_a/explains/bridges）
- **Checked**：按校验结果筛选（Pass/Fail/Conflict/All）
- **Confidence**：按置信度阈值过滤，便于从“全量探索图”切换到“高可信主干图”

# 6. 启动与运行（Quick Start）

## 6.1 一键启动

```
docker compose -p xkg up -d --build
```

- 前端UI：http://localhost:5173
- 后端API：http://localhost:8000

## 6.2 配置LLM模式

- 在线模式：LLM_PROVIDER=openai_compat 并配置 OPENAI_BASE_URL / OPENAI_API_KEY / OPENAI_MODEL 等（见 env.example）。

# 7. 分工说明

> 按照老师的课程建议，我们分成了算法与Agent负责人以及架构与部署负责人。

张辰阳：算法/Agent

- 设计并实现多阶段智能体：Planner/Retriever/Extractor/Bridge/Merge/Checker 工作流
- 后端API接口设计与前端框架搭建
- Check layer：证据一致性规则、关系降级策略、置信度更新、冲突标记

姚俊杰：架构与部署

- 前端细节、展示效果设计与优化
- 完成docker部署以及Dockerfile撰写
- Neo4j 存储层：约束创建、upsert_graph、子图查询与 expand 合并