技术规格书：YesBut v1.0

基于多智能体博弈与层状图网络的协同头脑风暴系统

1. 系统概述

1.1 定义

YesBut 是一个多智能体协同头脑风暴系统（Multi-Agent Collaborative Brainstorming System），用于将非结构化的想法或需求转化为结构化的、经过多方博弈验证的可执行方案。

核心理念：通过**发散-筛选-收敛**的三阶段流程，先大规模探索创意空间，再从中筛选可执行且有价值的方案，最后通过多智能体博弈收敛为可落地的决策方案。

三阶段流程：

1. **发散阶段（Divergence）**：大规模生成多样化创意，探索解空间的各个角落，包括非共识的激进方案
2. **筛选阶段（Filtering）**：基于可行性、价值和用户偏好，从大量创意中识别有潜力的候选方案
3. **收敛阶段（Convergence）**：对筛选出的候选方案进行多智能体博弈，通过辩证综合收敛为可执行方案

核心机制：

- 质量-多样性探索：在发散阶段采用 QD（Quality-Diversity）思想，同时优化方案质量和多样性
- 层状图网络：纵向树状分解 + 横向图状关联，支持深层递进推理
- 多智能体博弈：N 人非合作博弈模型，每个分支代表独立观点
- 双向推导：自顶向下演绎 + 自底向上归纳，形成逻辑闭环
- 语义熵验证：基于 NLI 模型检测 LLM 输出不确定性
- 贝叶斯偏好引出：通过信息增益最大化主动引出用户偏好

设计原则：

- 发散优先：在收敛之前充分探索解空间，避免过早陷入局部最优
- 事实与推理分离：所有事实必须通过外部验证，推理过程可追溯
- 博弈驱动收敛：通过 Nash 均衡求解多方冲突
- 用户偏好中心：效用函数基于用户偏好动态生成
- 可解释性优先：每个决策都有完整的证据链和置信度评估

1.2 设计理念

1.2.1 三阶段流程架构（Three-Phase Pipeline Architecture）

发散阶段（Divergence Phase）

- 目标：最大化解空间覆盖率，探索常规思维难以触及的创意角落
- 方法：采用质量-多样性（QD）算法思想，在多个特征维度上同时保留最优解
- 特征维度示例：风险等级、创新程度、实施周期、资源需求
- 生成策略：高温度采样、跨领域类比、约束松弛、逆向思维
- 输出：覆盖特征空间的大量候选方案（数十至数百个）
- 理论基础：MAP-Elites（Mouret & Clune, 2015）、发散思维（Guilford, 1967）

筛选阶段（Filtering Phase）

- 目标：从大量候选中识别可执行且有价值的方案
- 筛选维度：
  - 可行性（Feasibility）：是否在用户约束条件下可实现
  - 价值性（Value）：是否满足用户效用函数
  - 新颖性（Novelty）：是否提供非显而易见的洞察
- 方法：多目标 Pareto 筛选，保留非支配解集
- 输出：精简的候选方案集（5-15个）
- 理论基础：多目标优化（Deb et al., 2002）、Pareto 前沿

收敛阶段（Convergence Phase）

- 目标：通过多智能体博弈将候选方案收敛为可执行决策
- 方法：每个候选方案由一个分支管理智能体（BM）代言，进行辩论和综合
- 博弈机制：攻击/支持/综合，通过迭代最优响应逼近 Nash 均衡
- 输出：经过博弈验证的可执行方案，附带完整推理轨迹
- 理论基础：Nash 均衡（Nash, 1950）、论证框架（Dung, 1995）

1.2.2 多方博弈架构（N-Player Game Architecture）

- 系统采用 N 人非合作博弈模型（N-Player Non-Cooperative Game），而非二元对立的正反方辩论
- 每个分支管理智能体（BM）代表一个独立的观点持有者，持有异构效用函数
- 智能体间通过苏格拉底式诘问（Socratic Questioning）相互质疑，通过黑格尔辩证综合（Hegelian Dialectical Synthesis）整合冲突观点
- 理论基础：Nash 均衡（Nash, 1950）、论证框架（Argumentation Framework, Dung 1995）

1.2.2 层状图网络（Layered Graph Network）

- 纵向视图：树状结构，表示从母集到子集的层级分解关系
- 横向视图：图状结构，表示同一层级内节点间的支持、攻击、冲突关系
- 每一层推理必须在前一层的基础上进行更深层次的分析，禁止浮于表面的重复
- 理论基础：贝叶斯网络（Bayesian Network, Pearl 1988）、有向无环图（DAG）

1.2.3 用户双角色模型（Dual-Role User Model）

- 圆桌参会者（Roundtable Participant）：同步模式下，用户作为平等参与者加入讨论，可提出观点、质疑其他智能体、指定信息检索方向
- 顶层管理者（Executive Manager）：异步模式下，用户审阅智能体的离线工作成果，裁决不确定节点，设定约束条件
- 系统默认运行异步模式，除非用户主动切换至同步模式或发出停止指令

1.2.4 元议题分解（Atomic Topic Decomposition）

- 将初始想法或需求分解为不可再分的元议题（Atomic Topic）
- 元议题的重要性权重通过语义量化方法计算，支持精确的信息检索和资源分配
- 分解过程采用与多层推理相同的数学逻辑，确保一致性
- 理论基础：信息论（Information Theory, Shannon 1948）、层次分析法（AHP, Saaty 1980）

1.2.5 双向推导机制（Bidirectional Derivation）

- 自底向上（Bottom-Up）：从已知条件和检索到的事实出发，归纳推导可能的结论
- 自顶向下（Top-Down）：从最终需求出发，演绎分解所需的前置条件
- 双向推导在中间层相遇形成逻辑闭环，为智能体提供自主推理方向
- 理论基础：双向搜索（Bidirectional Search, Pohl 1971）、目标导向推理（Goal-Directed Reasoning）

1.2.6 可执行方案输出（Executable Plan Output）

- 输出不仅包含推理结论，还包含具体的执行方案和落地流程
- 系统询问用户当前条件，整合所需资源，生成针对用户具体情况的实施计划
- 方案拆分采用与元议题分解相同的逻辑，确保可操作性

1.3 功能概述

发散能力（Divergence Capabilities）

- 质量-多样性探索：在多个特征维度上同时保留最优解，避免过早收敛到局部最优
- 高温度创意生成：使用高随机性参数生成非常规方案
- 跨领域类比：从不相关领域引入概念，产生创新性碰撞
- 约束松弛探索：临时放宽约束条件，探索边界外的可能性
- 逆向思维生成：从反面或对立面出发生成方案
- 解空间可视化：将候选方案投射到特征空间，展示覆盖分布

筛选能力（Filtering Capabilities）

- 多目标 Pareto 筛选：保留非支配解集，平衡多个优化目标

- 可行性验证：基于用户约束条件和外部事实检验方案可执行性
- 价值评估：基于用户效用函数评估方案价值
- 新颖性检测：识别非显而易见的创新方案
- 盲区照亮：标识解空间中未被充分探索的区域

收敛能力（Convergence Capabilities）

- 层状图网络：纵向树状分解 + 横向图状关联，支持深层递进推理
- 双向推导：自顶向下演绎 + 自底向上归纳，形成逻辑闭环
- 元议题分解：细粒度拆分想法为语义可量化的原子单元
- 多分支并行推理：支持 N 个独立观点路径同时展开
- 双视图投影：因果视图（追溯推理路径）+ 冲突视图（识别互斥方案）

多方博弈机制

- N 人非合作博弈：每个分支代表独立观点，持有异构效用函数
- 苏格拉底式诘问：智能体间通过结构化质疑发现逻辑缺陷
- 黑格尔辩证综合：冲突观点通过综合节点整合为更高层次的方案
- Nash 均衡求解：多分支通过迭代最优响应收敛到稳定状态
- 攻击/支持机制：智能体间可相互质疑和验证，攻击需经审计验证

智能体协作

- 8 类专用智能体：
    - 创意生成智能体（Generator, GEN）：负责发散阶段的大规模创意生成
    - 需求解析智能体（RPA）：解析用户需求，引出偏好
    - 信息检索智能体（ISA）：从外部数据源获取事实
    - 审计合规智能体（ACA）：检测逻辑一致性和约束合规
    - 分支管理智能体（BM）：维护单个方案分支的推理过程
    - 调度仲裁智能体（GA）：管理资源分配和博弈协调
    - 效用量化智能体（UOA）：维护和更新效用函数
    - 输出编译智能体（REC）：将图结构转化为可执行方案
- 完全信息博弈：所有智能体共享全局知识库，不存在私有信息
- 大规模信息检索：通过 MCP 协议连接外部数据源，支持全面的事实收集

不确定性处理

- 语义熵验证：基于 NLI 模型的多次采样聚类，检测 LLM 输出不确定性
- 置信度评估：四维评估向量（有效性、效用、置信度、新颖性）
- 剪枝策略：自动删除、暂停、合并或折叠低质量节点
- 安全删除协议：考虑 Token 成本的渐进式节点清理机制
- 图论死锁熔断：检测并解除推理图中的循环依赖

效用函数量化

- 贝叶斯偏好引出：通过信息增益最大化的查询主动引出用户偏好

- 语义锚定量化：将抽象概念映射到可比较的数值区间
- 外部基准校准：通过 MCP 获取行业基准数据校准效用函数
- 多轮迭代修正：根据用户反馈持续优化效用函数参数

外部集成与第三方工具

信息收集架构

- 分支级独立检索：每个 BM 智能体独立为其负责的方案分支进行信息收集，避免信息同质化
- 批判驱动验证：当智能体间相互批判时，被攻击方必须提供外部证据支撑，攻击方必须提供反驳证据
- 交叉验证机制：同一事实主张需从多个独立数据源获取验证，降低单源偏差风险
- 证据链追溯：每个 FactNode 必须记录完整的数据来源 URI 和获取时间戳

搜索工具集成（通过 MCP 协议）

- Tavily Search：AI 原生搜索 API，返回结构化结果，适合深度研究
- Exa Search：语义搜索引擎，支持相似内容发现
- Brave Search API：隐私友好，独立索引
- Serper API：Google 搜索代理，低延迟
- SerpAPI：多搜索引擎聚合（Google、Bing、Yahoo、Baidu）

爬虫工具集成（通过 MCP 协议）

- Firecrawl：AI 友好的网页爬取，自动处理 JS 渲染，输出 Markdown/JSON
- Crawl4AI：专为 LLM 设计的开源爬虫，支持结构化数据提取
- Jina Reader：将任意网页转为 LLM 友好的 Markdown 格式
- Playwright MCP：浏览器自动化，处理需要登录或复杂交互的页面

学术与专业数据源

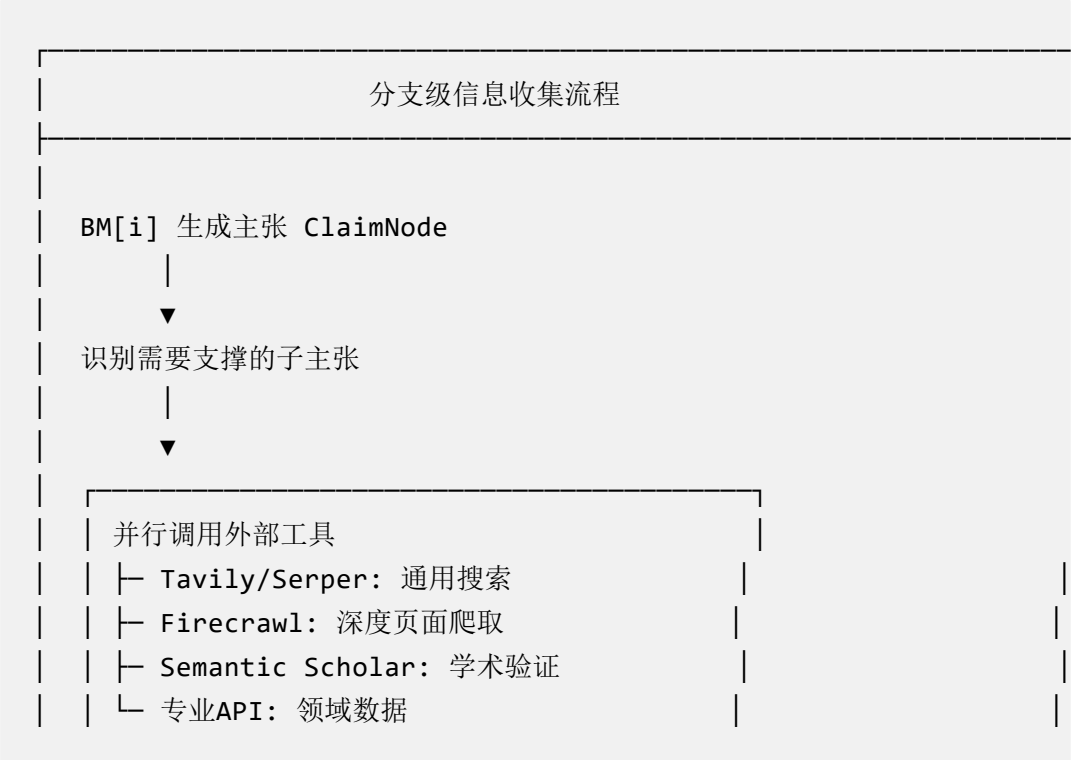
- Semantic Scholar API：学术论文检索和引用分析
- arXiv API：预印本论文获取
- PubMed API：生物医学文献
- USPTO/EPO API：专利检索
- SEC EDGAR：美国上市公司财务数据
- 企业信息 API：天眼查、企查查等

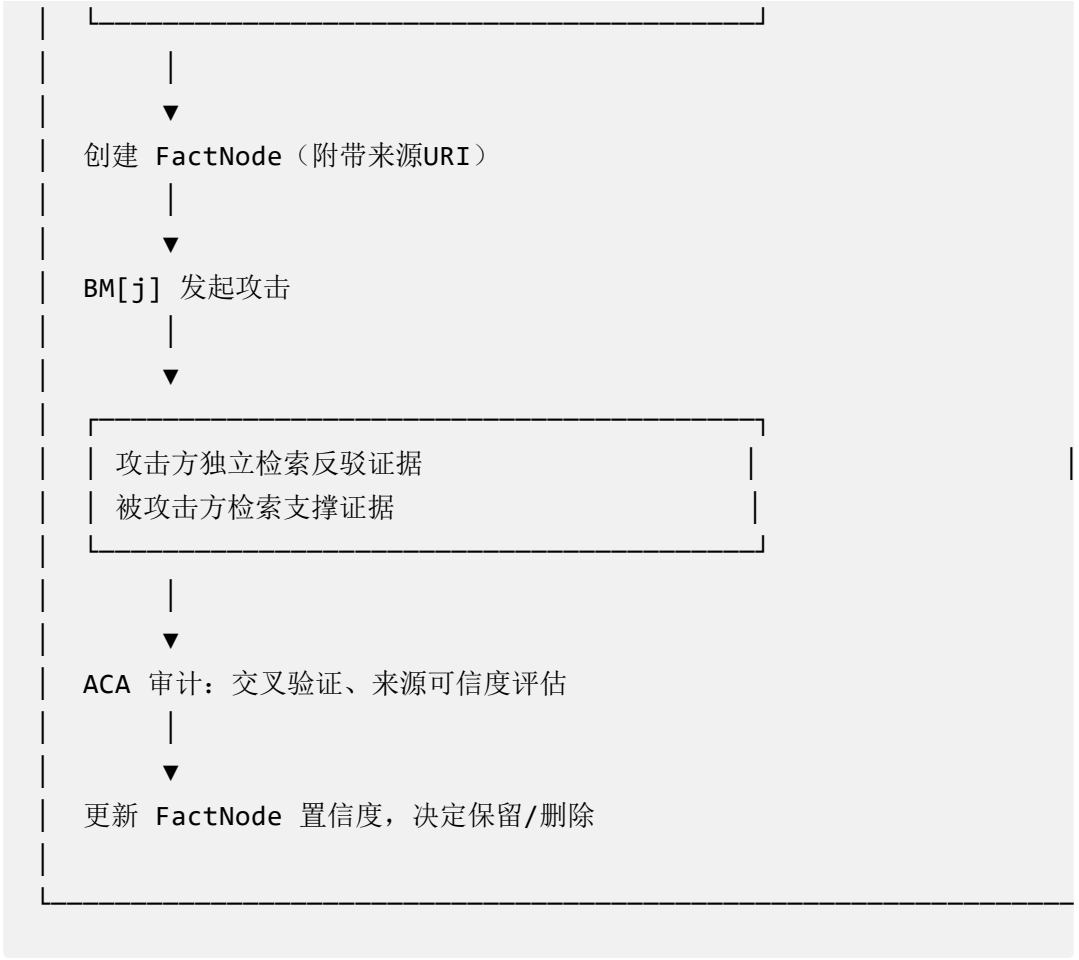
核心功能 vs 外包功能

功能类别	自研（核心）	外包（第三方）
推理编排	三阶段流程控制、分支管理、博弈协调	-

信息检索	检索策略、查询生成、结果整合	搜索API（Tavily/Serper）、爬虫（Firecrawl）
事实验证	语义熵计算、交叉验证逻辑	NLI模型（HuggingFace）、事实核查API
向量存储	检索策略、相似度阈值	向量数据库（Pinecone/Weaviate/Milvus）
知识图谱	实体关系抽取逻辑、路径查询	图数据库（Neo4j/Amazon Neptune）
代码执行	执行策略、安全边界	沙箱环境（E2B/Modal）
文档解析	解析策略、结构化映射	解析工具（Unstructured/LlamaParse）
嵌入计算	嵌入策略、缓存管理	嵌入模型（OpenAI/Cohere/Voyage）
LLM推理	提示词设计、温度调度、输出解析	LLM API（Claude/GPT-4/Gemini）
数学计算	计算策略、结果验证	计算引擎（Wolfram Alpha/SymPy）

分支级信息收集流程





MCP 服务器配置示例

```
{
  "mcpServers": {
    "tavily": {
      "command": "npx",
      "args": ["-y", "tavily-mcp"],
      "env": {"TAVILY_API_KEY": "<key>"}
    },
    "firecrawl": {
      "command": "npx",
      "args": ["-y", "firecrawl-mcp"],
      "env": {"FIRECRAWL_API_KEY": "<key>"}
    },
    "brave-search": {
      "command": "npx",
      "args": ["-y", "@anthropic/mcp-server-brave-search"],
      "env": {"BRAVE_API_KEY": "<key>"}
    },
```

```
    "playwright": {
      "command": "npx",
      "args": ["-y", "@anthropic/mcp-server-playwright"]
    }
  }
}
```

交互模式

- 同步模式（圆桌参会者）：用户实时参与讨论，提出观点，质疑智能体，指定检索方向
- 异步模式（顶层管理者）：系统后台运行，用户审阅成果，裁决不确定节点
- 默认异步：除非用户主动切换或发出停止指令，系统持续异步运行
- 深度研究模式：支持长周期（30分钟至数小时）的后台研究任务，包括大规模文献检索、知识图谱构建、多轮模拟推演，最终交付结构化研究报告
- 空间画布视图：可选的二维可视化界面，将推理图投射到空间平面，支持节点拖拽、聚类查看、热力图叠加，辅助用户理解复杂决策结构

自优化机制

- TextGrad 提示词优化：基于文本梯度迭代改进智能体提示词
- 冻结策略：核心智能体（ACA、GA、RPA）保持稳定，仅优化推理智能体
- 版本控制与回滚：保留历史版本，性能下降时自动回滚

输出能力

- 结构化输出：Markdown/JSON 格式的行动计划
- 推理轨迹：完整的证据链和置信度评估
- 综合方案生成：冲突分支的优势合并
- 可执行落地方案：询问用户条件，生成具体实施计划
- 解空间多样性展示：在收敛到最终方案前，展示候选方案在多个维度（如风险/收益、短期/长期）上的分布，帮助用户理解决策边界和被放弃的替代方案

1.4 核心问题域

本系统解决以下技术问题：

1. **状态空间爆炸**：复杂决策问题的搜索空间呈指数增长，需要有效的剪枝策略。
2. **LLM 幻觉**：大语言模型生成的内容可能与事实不符，需要外部验证机制。
3. **多目标优化**：决策涉及多个相互冲突的目标，需要 Pareto 最优解。
4. **偏好不确定性**：用户偏好通常是隐式的、不完整的，需要主动引出。
5. **效用函数量化困难**：抽象概念难以精确量化，需要语义锚定和外部校准机制。
6. **推理深度不足**：表面化的分析无法产生有价值的洞察，需要强制深层递进机制。

1.5 核心技术组件

- **层状图网络（Layered Graph Network）**：纵向为树状层级分解结构，横向为图状关联结构。节点和边可在运行时增删，支持增量式图构建。推理路径组织为分支（Branch），每个分支是一条从根节点到叶节点的路径。
- **语义熵（Semantic Entropy）**：基于自然语言推理（NLI）模型的多次采样聚类，用于检测 LLM 输出的不确定性（参考：Kuhn et al., ICLR 2023）。
- **Shapley 值计算**：利用 DAG 拓扑结构优化合作博弈论中 Shapley 值的计算复杂度。
- **贝叶斯偏好引出**：通过贝叶斯优化选择信息增益最大的查询，主动引出用户偏好。
- **语义锚定量化（Semantic Anchoring Quantification）**：将抽象语义概念映射到数值区间的方法。
- **JSON Schema 数据规范**：标准化的节点、边、分支对象序列化格式。

---

2. 图结构定义

2.1 层状图网络形式化定义

定义：层状图网络 $G = (L, V, E^v, E^h)$，其中：

- $L = \{l^0, l^1, ..., l^k\}$：层级集合，$l^0$ 为根层，l^k 为叶层
- $V = \bigcup_{i=0}^{k} V^i$：节点集合，$V^i$ 为第 i 层的节点集
- $E^v \subseteq \bigcup_{i=0}^{k-1} (V^i \times V^{i+1})$：纵向边集合，表示层级间的父子关系
- $E^h \subseteq \bigcup_{i=0}^{k} (V^i \times V^i)$：横向边集合，表示同层节点间的关系

纵向视图（树状结构）：

- 投影 $G^v = (V, E^v)$ 形成森林结构
- 每个节点最多有一个父节点
- 从母集到子集的分解关系

横向视图（图状结构）：

- 投影 $G_h^i = (V^i, E^h \cap (V^i \times V^i))$ 形成同层关联图
- 节点间可存在支持、攻击、冲突关系
- 用于博弈分析和冲突检测

2.2 节点类型

系统定义以下节点类型：

类型	标识符	描述	可变性	层级约束

目标节点	`GoalNode`	最终决策目标或初始想法	用户可修改	仅存在于 ℓ^0
约束节点	`ConstraintNode`	用户定义的硬约束条件	用户可修改	可存在于任意层
主张节点	`ClaimNode`	智能体生成的推理结论	系统可修改	可存在于任意层
事实节点	`FactNode`	经外部验证的客观事实	不可变	通常存在于叶层
元议题节点	`AtomicTopicNode`	不可再分的原子议题	系统可修改	分解过程的终点
待验证节点	`PendingNode`	自顶向下生成的待匹配节点	系统可修改	等待自底向上匹配

2.3 边类型

系统定义以下边类型：

纵向边（Vertical Edge）：

类型	语义	方向
`decompose`	父节点分解为子节点	父 → 子
`derive`	子节点推导出父节点	子 → 父

横向边（Horizontal Edge）：

类型	语义	权重范围
`support`	源节点为目标节点提供正向证据	$[0, 1]$
`attack`	源节点削弱目标节点的可信度	$[0, 1]$
`conflict`	两节点互斥，不能同时为真	$\{0, 1\}$
`entail`	源节点逻辑蕴含目标节点	$\{0, 1\}$

2.4 超边定义

超边用于表示多个节点之间的复杂关系：

- **冲突超边（Conflict Hyperedge）**：连接两个或多个互斥的 `ClaimNode`，表示这些节点不能同时为真。

- **依赖超边（Dependency Hyperedge）**：连接一组节点，表示目标节点依赖于所有源节点。
- **聚合超边（Aggregation Hyperedge）**：连接多个节点到一个综合节点，表示信息聚合关系。

2.5 分支结构

定义：分支 $B = (r, P, L, u)$，其中：

- r：根节点（通常是 `GoalNode` 或初始 `ClaimNode`）
- P：从根到叶的路径集合
- L：叶节点集合（当前推理前沿）
- u：该分支绑定的效用函数

分支操作：

- `fork(B, n)`：在节点 n 处分叉，创建新分支，继承父分支的效用函数或生成新效用函数
- `merge(B1, B2)`：合并两个分支，创建综合节点，效用函数取加权平均
- `prune(B)`：删除分支及其所有节点，触发安全删除协议

2.6 双视图投影

为支持不同的分析任务，系统提供两种图投影：

2.6.1 因果视图（Causal View）

- **结构**：有向无环图（DAG）
- **用途**：追溯推理路径，审查逻辑链完整性，验证推理深度
- **投影规则**：保留所有纵向边 E_v 和横向边中的 `support`、`entail` 类型

2.6.2 冲突视图（Conflict View）

- **结构**：无向图
- **用途**：识别互斥方案，计算博弈均衡，检测死锁
- **投影规则**：仅保留 `attack`、`conflict` 类型的边和冲突超边

2.7 层级深度约束

为确保推理深度，系统强制执行以下约束：

深度递进规则：

- 子节点的语义粒度必须小于父节点
- 子节点必须包含父节点未明确表达的新信息
- 禁止同义重复：若 $\text{sim}(v^{child}, v^{parent}) > 0.9$，拒绝创建该子节点

最小深度要求：

- 对于复杂议题，推理深度 $k \geq 3$
- 每层至少产生 2 个有效子节点

---

3. 智能体架构

3.1 系统架构

系统采用分层架构，包含四个功能平面：

3.2 智能体定义

3.2.1 需求解析智能体 (Requirement Parsing Agent, RPA)

职责：将自然语言需求转化为形式化约束集合。

输入：用户自然语言描述 **输出**：约束集合 $C = \{c^1, c^2, ..., c^n\}$，其中每个 c^i 为硬约束或软约束

核心算法：贝叶斯偏好引出

1. 初始化先验分布 $P(\theta)$，其中 θ 为用户偏好参数
2. 选择信息增益最大的查询 $q^* = \arg\max_q I(q; \theta)$
3. 获取用户响应，更新后验 $P(\theta|response)$
4. 重复直至后验方差低于阈值

效用函数合成：基于引出的偏好参数，生成可执行的效用函数代码：

```
def utility(plan: Plan, params: PreferenceParams) -> float:
    score = sum(params.weights[i] * plan.metrics[i]
                for i in range(len(params.weights)))
    for constraint in params.hard_constraints:
        if not constraint.satisfied(plan):
            return float('-inf')
    return score
```

3.2.2 信息检索智能体 (Information Scout Agent, ISA)

职责：从外部数据源获取事实信息，构建 `FactNode`。

核心原则：禁止使用 LLM 内部知识作为事实来源，所有事实必须通过 MCP 协议从外部验证。

MCP（Model Context Protocol）集成：

- MCP 是 Anthropic 开发的开源协议，用于标准化 AI 应用与外部系统的连接
- 支持的数据源类型：Web Search API、学术数据库、向量数据库、代码执行沙箱

语义熵验证流程：

1. 对同一查询执行 K 次独立检索（$K \geq 5$）
2. 使用 NLI 模型计算检索结果之间的语义等价性
3. 将语义等价的结果聚类
4. 计算语义熵：$H^{sem} = -\sum_i p^i \log p^i$，其中 p^i 为第 i 个聚类的比例
5. 若 $H^{sem} > \epsilon$（阈值），标记为高不确定性，触发深度检索

3.2.3 审计智能体 (Audit & Compliance Agent, ACA)

职责：检测系统输入输出的逻辑一致性。

核心算法：一阶逻辑矛盾检测

- 将节点内容转化为一阶逻辑命题
- 使用 SAT 求解器检测命题集合的可满足性
- 若不可满足，定位矛盾源并生成 `attack` 边

3.2.4 分支管理智能体 (Branch Manager Agent, BM)

职责：维护单个分支的推理过程。

核心算法：状态空间搜索

- 状态：当前分支的节点集合和边集合
- 动作：`{add_node, add_edge, query_ISA, attack_node}`
- 转移：基于 LLM 生成的推理步骤
- 终止：达到目标节点或无法继续扩展

3.2.5 调度智能体 (Game Arbiter, GA)

职责：管理多分支间的资源分配和博弈协调。

核心算法：qEHVI（q-Expected Hypervolume Improvement）

- qEHVI 是多目标贝叶斯优化算法（Daulton et al., NeurIPS 2020）
- 用于在多个分支间分配计算资源
- 目标：最大化 Pareto 前沿的超体积增量

分支相似度计算：

- 使用嵌入模型计算分支叶节点的向量表示
- 相似度：$Sim(B^1, B^2) = \cos(\vec{v}_{B^1}, \vec{v}_{B^2})$
- 若 $Sim < \theta$，视为独立分支，分配独立 BM

3.2.6 效用量化智能体 (Utility Optimization Agent, UOA)

职责：维护和更新效用函数。

交互式优化：当用户对系统输出不满意时，UOA 触发新一轮偏好引出，更新效用函数参数。

3.2.7 输出编译智能体 (Reverse Engineering Compiler, REC)

职责：将图结构转化为可执行的行动计划。

算法：拓扑排序 + 模板填充

1. 对因果视图执行拓扑排序
2. 按排序顺序遍历节点
3. 为每个 `ClaimNode` 生成对应的行动步骤
4. 输出结构化文档（Markdown/JSON）

3.3 节点评估与剪枝策略

为控制状态空间规模，系统对每个节点 v 计算四维评估向量 $\vec{E}(v) = [S^{val}, S^{util}, S^{conf}, S^{nov}]$。

3.3.1 逻辑有效性 (S^{val})

定义：节点在论证框架（Argumentation Framework, Dung 1995）中的接受状态。

计算规则：

- 若存在 `FactNode` f 使得 $(f, v) \in E^{attack}$ 且 f 已验证，则 $S^{val} = 0$
- 否则 $S^{val} = 1$

处理：$S^{val} = 0$ 的节点立即删除，包括其所有后继节点。

3.3.2 边际效用 (S^{util})

定义：节点对最终目标的贡献度，基于 Shapley 值计算。

Shapley 值定义：

$$\phi^i(v) = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(|N| - |S| - 1)!}{|N|!} [v(S \cup \{i\}) - v(S)]$$

DAG 优化：利用图的拓扑结构，仅计算节点祖先集合的子集，将复杂度从 $O(2^n)$ 降至 $O(n \cdot d)$，其中 d 为平均入度。

处理：$S^{util} < \delta$ 的节点标记为 `collapsed`，在默认视图中隐藏。

3.3.3 置信度 (S^{conf})

定义：节点内容为真的概率估计。

计算：$S^{conf} = 1 - H^{sem}(v)$，其中 H^{sem} 为语义熵。

处理：$S^{conf} < \epsilon$ 的节点标记为 `unstable`，暂停后继节点生成，等待 ISA 补充证据。

3.3.4 新颖性 (S^{nov})

定义：节点与现有节点集的语义差异度。

计算：使用局部敏感哈希（LSH）进行近似最近邻搜索：

1. 计算节点嵌入向量 $\vec{v}$
2. 使用 LSH 查找 k 个最近邻
3. $S^{nov} = 1 - \max^{u \in kNN} \cos(\vec{v}, \vec{u})$

处理：$S^{nov} < \theta$ 的节点与最相似节点合并，保留 S^{util} 较高者。

3.3.5 剪枝决策函数

```
function prune_decision(node v):
    if S_val(v) == 0:
        return DELETE
    if S_conf(v) < epsilon:
        return SUSPEND
    if S_nov(v) < theta:
        return MERGE
    if S_util(v) < delta:
        return COLLAPSE
    return KEEP
```

---

4. 多智能体博弈机制

4.1 博弈形式化定义

系统中的多分支推理建模为 N 人非合作博弈 $G = (N, A, u)$：

- **参与者集合** $N = \{BM^1, BM^2, ..., BM^n\}$：每个分支管理智能体为一个参与者，代表一个独立观点
- **动作集合** $A^i = \{query, attack, support, merge, question\}$：每个参与者的可选动作
- **效用函数** $u^i : A \to \mathbb{R}$：参与者 i 的异构效用函数

4.2 信息结构

完全信息假设：所有智能体共享全局知识库，不存在私有信息。

异构效用函数：虽然信息对称，但不同分支的效用函数不同：

- 每个分支绑定一个效用函数 u^i
- 效用函数由 UOA 根据用户偏好和分支特性生成
- 不同效用函数对同一方案的评估可能不同，这是多方博弈的基础

4.3 苏格拉底式诘问机制（Socratic Questioning Mechanism）

定义：智能体通过结构化的质疑序列，暴露其他智能体论证中的逻辑缺陷或隐含假设。

诘问类型：

类型	目的	示例模板
澄清性诘问	要求明确定义或范围	"节点 X 中的 [术语] 具体指什么？"
假设性诘问	暴露隐含前提	"节点 X 的结论是否依赖于 [假设]？"
证据性诘问	要求提供支撑证据	"节点 X 的主张基于什么证据？"
反例性诘问	提出反驳案例	"如果 [条件]，节点 X 的结论是否仍然成立？"
后果性诘问	追问逻辑推论	"如果节点 X 成立，是否意味着 [后果]？"

诘问流程：

1. BM^i 选择目标节点 v^j
2. BM^i 生成诘问节点 v^q，类型为上述之一
3. BM^j 必须响应诘问，生成回应节点 v^r
4. ACA 评估回应的充分性
5. 若回应不充分，v^j 的置信度 S^{conf} 降低

4.4 攻击机制

攻击定义：智能体 BM^i 对节点 v^j（属于分支 B^j）发起攻击，当且仅当：

1. BM^i 识别出 v^j 的逻辑缺陷（推理错误、证据不足、与事实矛盾）
2. BM^i 生成攻击节点 v^{attack} 和攻击边 (v^{attack}, v^j)
3. ACA 验证攻击的有效性

攻击效果：成功的攻击降低目标节点的置信度 S^{conf}。

攻击与诘问的区别：

- 诘问是探索性的，要求对方澄清或补充
- 攻击是断言性的，声称对方存在错误

4.5 黑格尔辩证综合机制（Hegelian Dialectical Synthesis）

定义：当多个分支存在冲突但各自包含有效成分时，系统通过辩证综合生成更高层次的方案。

三阶段过程：

1. **正题（Thesis）**：分支 B^i 提出主张 v^i
2. **反题（Antithesis）**：分支 B^j 提出与 v^i 冲突的主张 v^j

3. **合题（Synthesis）**：系统生成综合节点 v^s，满足：
    - 保留 v^i 和 v^j 中与 `FactNode` 一致的部分
    - 解决 v^i 和 v^j 之间的矛盾
    - 不违反任何 `ConstraintNode`

综合节点生成算法：

1. 识别冲突超边连接的节点集合 $\{v^1, v^2, ..., v^m\}$
2. 提取每个节点的核心主张和支撑证据
3. 识别主张间的共同点和分歧点
4. 使用 LLM 生成综合方案，指令包含：
    - 必须保留的共同点
    - 需要调和的分歧点
    - 不可违反的约束条件
5. ACA 验证综合方案的逻辑一致性
6. 若验证失败，迭代修正直至通过

4.6 均衡求解

目标：寻找 Nash 均衡，即没有任何参与者可以通过单方面改变策略来提高自身效用的状态。

Nash 均衡定义：策略组合 $s^* = (s^{*1}, ..., s^{*n})$ 是 Nash 均衡，当且仅当：

$$\forall i, \forall s^i \neq s^{*i} : u^i(s^{*i}, s^{*-i}) \geq u^i(s^i, s^{*-i})$$

求解方法：

1. 构建冲突视图，识别互斥分支
2. 计算每个分支的效用值
3. 使用迭代最优响应（Iterated Best Response）逼近均衡
4. 若存在多个均衡，选择 Pareto 最优的均衡
5. 若无法达到均衡，触发辩证综合机制

---

5. 推理流程

5.1 元议题分解（Atomic Topic Decomposition）

目标：将用户输入的想法或需求分解为不可再分的元议题，每个元议题具有可量化的语义权重。

原子性判定条件：

- 不包含逻辑连接词（且、或、如果...则）
- 不包含多个独立的评估维度

- 可以通过单一信息检索任务获取相关事实
- 可以用单一效用函数维度进行评估

分解算法：

1. 使用 LLM 将输入文本分解为候选命题集合 $S = \{s^1, s^2, ..., s^n\}$
2. 对每个 s^i 执行原子性检验：
    - 若包含逻辑连接词，递归分解
    - 若包含多个评估维度，按维度拆分
3. 验证分解的完备性：$\bigcup_i s^i$ 应覆盖原始输入的所有语义
4. 为每个 s^i 创建 `AtomicTopicNode`

语义权重计算：

- **信息熵权重**：$w^{entropy}(s^i) = H(s^i) / \sum_j H(s^j)$，其中 $H(s^i)$ 为该议题的信息熵
- **用户偏好权重**：$w^{pref}(s^i)$ 由 UOA 通过贝叶斯偏好引出获取
- **依赖度权重**：$w^{dep}(s^i) = |\{s^j : s^j \text{ depends on } s^i\}| / n$
- **综合权重**：$w(s^i) = \alpha \cdot w^{entropy}(s^i) + \beta \cdot w^{pref}(s^i) + \gamma \cdot w^{dep}(s^i)$

理论基础：

- 信息论（Information Theory, Shannon 1948）：信息熵衡量不确定性
- 层次分析法（Analytic Hierarchy Process, Saaty 1980）：多准则决策分解

5.2 自底向上推理（Bottom-Up Reasoning）

路径：`FactNode` → `ClaimNode` → `GoalNode` **语义**：从已知条件和检索到的事实出发，归纳推导可能的结论

流程：

1. ISA 根据元议题权重优先级，从外部数据源获取事实，创建 `FactNode`
2. BM 基于 `FactNode` 生成假设性 `ClaimNode`
3. 计算条件概率 $P(Claim|Evidence)$
4. 若 $P > \tau$（阈值），创建 `derive` 边并继续向上推理
5. 每次向上推理必须增加抽象层级，禁止同层重复

5.3 自顶向下推理（Top-Down Reasoning）

路径：`GoalNode` → `ConstraintNode` → `PendingNode` **语义**：从最终需求出发，演绎分解所需的前置条件

流程：

1. 从 `GoalNode` 出发，识别实现目标的必要条件
2. 为每个必要条件创建 `PendingNode`（待验证节点）

3. PendingNode 等待自底向上推理的匹配
4. 使用目标导向搜索：优先展开与 PendingNode 语义相关的分支

5.4 双向匹配与逻辑闭环

定义：当自顶向下生成的 PendingNode 与自底向上生成的 ClaimNode 语义匹配时，形成逻辑闭环。

匹配算法：

1. 计算 PendingNode 和候选 ClaimNode 的嵌入向量
2. 若余弦相似度 > 0.9，触发匹配验证
3. 使用 NLI 模型确认语义蕴含关系
4. 匹配成功后：
    - 将 PendingNode 转换为 ClaimNode
    - 路径上所有节点的 S^{conf} 提升
    - 创建 entail 边连接匹配的节点

信息缺口处理：

- 若 PendingNode 长时间无法匹配，系统生成信息检索任务
- ISA 主动搜索相关证据
- 若检索失败，标记为"信息缺口"，提交用户决策

闭环的意义：

- 证明推理路径的完整性
- 为智能体提供自主推理方向
- 在无人类监管时持续收集信息和推理

5.5 温度调度策略

系统使用温度参数 T 控制 LLM 生成的随机性，实现探索-利用平衡。

调度函数：$T(t) = T^0 \cdot \exp(-\lambda t)$，其中 t 为迭代次数。

阶段划分：

阶段	温度范围	行为特征
探索期	$T \in [0.7, 1.0]$	生成多样化假设，容忍不完善方案
收敛期	$T \in [0.3, 0.7]$	聚焦于完善现有方案，减少新分支
验证期	$T \in [0.0, 0.3]$	严格逻辑检查，剪枝无法闭环的路径

---

6. 资源调度与计算分工

6.1 多目标资源调度

问题：在有限计算资源（Token 预算、时间限制）下，如何分配资源给多个分支？

算法：qEHVI（q-Expected Hypervolume Improvement）

qEHVI 是多目标贝叶斯优化算法，用于在多个候选点中选择最优的 q 个点进行评估。

应用于分支调度：

1. 将每个分支视为候选点
2. 目标向量：$[S^{util}, S^{conf}, -cost]$（效用、置信度、负成本）
3. 计算每个分支的期望超体积改进
4. 选择 qEHVI 最大的分支分配计算资源

超体积定义：给定 Pareto 前沿 P 和参考点 r，超体积为 P 支配的区域体积：

$$HV(P, r) = \lambda(\{y \in \mathbb{R}^m : \exists p \in P, r \prec y \prec p\})$$

6.2 LLM 与确定性算法的职责边界

任务类型	执行者	理由
语义相似度判断	LLM (NLI)	需要语言理解能力
假设生成	LLM	需要创造性推理
代码生成	LLM	需要语言到代码的转换
概率计算	确定性算法	需要精确数值计算
图遍历	确定性算法	需要保证正确性
Shapley 值计算	确定性算法	需要精确博弈论计算
拓扑排序	确定性算法	需要保证正确性

6.3 计算复杂度分析

操作	时间复杂度	空间复杂度
节点添加	$O(1)$	$O(1)$
边添加	$O(1)$	$O(1)$
剪枝决策	$O(k)$，k 为近邻数	$O(1)$
Shapley 值（DAG 优化）	$O(n \cdot d)$	$O(n)$

qEHVI 计算	$O(q \cdot m \cdot	P
语义熵计算	$O(K^2)$，K 为采样数	$O(K)$

6.4 效用函数量化机制（Utility Function Quantification）

问题：抽象概念（如"创新性"、"可行性"、"风险"）难以精确量化，导致效用函数设计不准确，进而影响 Pareto 优化和 qEHVI 的有效性。

解决方案：语义锚定量化（Semantic Anchoring Quantification）

6.4.1 锚点定义 为每个抽象维度定义语义锚点（Semantic Anchor），将连续数值区间映射到具体的语义描述：

数值区间	语义锚点示例（以"可行性"为例）
$[0.0, 0.2]$	技术上不可能或需要突破性创新
$(0.2, 0.4]$	技术可行但需要大量研发投入
$(0.4, 0.6]$	技术成熟但实施存在挑战
$(0.6, 0.8]$	可直接实施，风险可控
$(0.8, 1.0]$	已有成熟方案，可立即执行

6.4.2 外部基准校准 通过 MCP 协议获取外部基准数据，校准效用函数参数：

- 行业标准：从行业报告、学术文献获取基准值
- 历史数据：从用户历史决策中学习偏好模式
- 市场数据：从市场调研、竞品分析获取参考值

6.4.3 多轮迭代修正

1. 初始化：LLM 基于语义锚点生成初始效用函数
2. 校准：使用外部基准数据调整参数
3. 验证：向用户展示典型案例的评估结果
4. 修正：根据用户反馈调整权重和锚点定义
5. 收敛：当用户满意度达到阈值时停止迭代

6.4.4 不确定性传播 效用函数的不确定性通过以下方式传播到最终决策：

- 每个效用值附带置信区间 $[u - \sigma, u + \sigma]$
- Pareto 优化考虑置信区间的重叠
- 高不确定性的维度触发额外的信息检索

理论基础：

- 锚定效应（Anchoring Effect, Tversky & Kahneman 1974）：认知心理学中的判断偏差理论
- 多属性效用理论（Multi-Attribute Utility Theory, Keeney & Raiffa 1976）：决策分析的数学基础

6.5 安全删除协议（Safe Deletion Protocol）

问题：节点删除可能导致信息丢失，且频繁删除消耗大量 Token 用于重建。

设计原则：

- 渐进式清理：优先软删除，延迟硬删除
- 成本感知：考虑重建成本与存储成本的权衡
- 可恢复性：保留删除历史，支持回滚

6.5.1 删除状态机

```
ACTIVE → COLLAPSED → SUSPENDED → SOFT_DELETED → HARD_DELETED
```

状态	可见性	可恢复性	Token 成本
ACTIVE	完全可见	N/A	0
COLLAPSED	默认隐藏，可展开	立即恢复	0
SUSPENDED	不可见，保留数据	立即恢复	0
SOFT_DELETED	不可见，保留摘要	需重建部分内容	低
HARD_DELETED	不可见，无数据	需完全重建	高

6.5.2 删除决策函数

```
function deletion_decision(node v, budget B):
    rebuild_cost = estimate_rebuild_cost(v)
    storage_cost = estimate_storage_cost(v)
    utility_loss = estimate_utility_loss(v)

    if utility_loss > threshold_high:
        return KEEP
    if rebuild_cost > B * 0.1:  // 重建成本超过预算10%
        return COLLAPSED
    if storage_cost < rebuild_cost * 0.5:
        return SUSPENDED
    if age(v) < 24h:
```

```
        return SOFT_DELETED
    return HARD_DELETED
```

6.5.3 批量清理策略

- 定期执行清理任务，而非实时删除
- 按层级从叶节点向根节点清理
- 保留高 Shapley 值节点的完整数据

6.6 图论死锁熔断机制（Graph Deadlock Circuit Breaker）

问题：推理图中可能出现循环依赖，导致系统无法继续推进。

死锁类型：

类型	描述	示例
循环依赖	节点 A 依赖 B，B 依赖 A	A 需要 B 的结论，B 需要 A 的结论
资源竞争	多个分支竞争同一信息源	多个 BM 同时请求同一 MCP 资源
等待链	多个 PendingNode 形成等待环	A 等待 B，B 等待 C，C 等待 A

6.6.1 死锁检测

- 周期性执行环检测算法（Cycle Detection）
- 检测频率：每 N 次节点操作后执行一次
- 算法：深度优先搜索（DFS）标记法

6.6.2 熔断策略

```
function circuit_breaker(cycle C):
    // 1. 识别环中最弱节点
    weakest = argmin_{v in C} S_conf(v)

    // 2. 尝试外部解决
    if can_resolve_externally(weakest):
        trigger_ISA_search(weakest)
        return RESOLVED

    // 3. 强制打破环
    if cycle_age(C) > timeout:
        mark_as_unresolvable(weakest)
        create_user_decision_request(C)
        return ESCALATED
```

```
    // 4. 等待更多信息
    return WAITING
```

6.6.3 熔断后处理

- 记录死锁事件，用于系统优化
- 通知用户死锁原因和影响范围
- 提供手动解决选项

理论基础：

- 死锁检测（Deadlock Detection, Coffman et al. 1971）：操作系统中的经典问题
- 熔断器模式（Circuit Breaker Pattern, Nygard 2007）：分布式系统的容错设计

---

7. 用户交互模式

7.1 用户双角色模型

7.1.1 圆桌参会者（Roundtable Participant）- 同步模式

- **角色定位**：用户作为平等参与者加入讨论
- **权限**：
  - 提出观点（创建 `ClaimNode`）
  - 质疑智能体（发起诘问）
  - 指定信息检索方向（指派 ISA 任务）
  - 支持或攻击其他节点
- **约束**：用户输入需经 ACA 一致性检查

7.1.2 顶层管理者（Executive Manager）- 异步模式

- **角色定位**：用户审阅智能体的离线工作成果
- **权限**：
  - 设定约束条件（创建 `ConstraintNode`，无需审计）
  - 裁决不确定节点
  - 批准或否决综合方案
  - 调整效用函数参数
- **约束**：无

7.2 同步模式详解

适用场景：实时交互式头脑风暴会议

交互流程：

1. 用户提交输入（观点、质疑、检索指令）
2. ACA 对用户输入执行一致性检查
3. 若检测到与现有 `FactNode` 或 `ConstraintNode` 的冲突，系统请求用户提供证据或修改输入
4. 通过检查的输入创建为相应节点类型，进入正常推理流程
5. 用户可随时查看当前推理图状态
6. 用户可指定下一步探索方向

用户操作权限：

操作	是否需要审计	说明
添加 `ClaimNode`	是	用户观点需验证一致性
添加 `ConstraintNode`	否	用户权威，直接生效
发起诘问	否	诘问是探索性的
发起攻击	是	攻击需验证有效性
删除节点	是	防止误删重要节点
指派 ISA 任务	否	用户可自由指定检索方向

7.3 异步模式详解

适用场景：长时间运行的深度分析任务 **默认行为**：系统默认运行异步模式，除非用户主动切换至同步模式或发出停止指令

系统行为：

1. 系统在后台持续运行推理流程
2. 智能体自主进行信息检索、推理、博弈
3. 当遇到以下情况时，暂停并等待用户决策：
    - 置信度接近 0.5 的关键节点（高不确定性）
    - 效用值相近的互斥方案（需要偏好判断）
    - 检测到潜在的高风险决策
    - 信息缺口无法通过检索填补
    - 死锁熔断触发

待决队列：

```
{
  "pending_decisions": [
    {
      "node_id": "node_xyz",
```

```
      "reason": "low_confidence",
      "confidence": 0.52,
      "context": "关于市场规模的估算存在分歧",
      "options": ["approve", "reject", "request_more_evidence"],
      "deadline": "2025-01-20T10:00:00Z"
    }
  ]
}
```

用户决策传播：用户对待决节点的决策作为强监督信号，更新该节点及其所有后继节点的状态。

7.4 模式切换

- **异步 → 同步**：用户发送"加入讨论"指令
- **同步 → 异步**：用户发送"离开讨论"指令或超时无响应
- **紧急中断**：用户发送"停止"指令，系统保存当前状态并暂停

---

8. 数据结构规范

8.1 节点对象 (Node)

```
{
  "id": "string (UUID)",
  "type": "enum: fact | constraint | claim | goal",
  "branch_id": "string",
  "content": "string",
  "evaluation": {
    "validity": "number [0, 1]",
    "utility": "number [0, 1]",
    "confidence": "number [0, 1]",
    "novelty": "number [0, 1]"
  },
  "provenance": {
    "source_agent": "string",
    "mcp_uri": "string[] (optional)",
    "created_at": "ISO 8601 timestamp"
  },
  "status": "enum: active | collapsed | suspended | deleted"
}
```

8.2 边对象 (Edge)

```
{
  "id": "string (UUID)",
  "source": "string (node_id)",
  "target": "string (node_id)",
  "type": "enum: support | attack | entail",
  "weight": "number [0, 1]",
  "created_by": "string (agent_id)"
}
```

8.3 超边对象 (Hyperedge)

```
{
  "id": "string (UUID)",
  "nodes": "string[] (node_ids, length >= 2)",
  "type": "enum: conflict | dependency | aggregation",
  "metadata": {
    "semantic_distance": "number (optional)",
    "resolution_status": "enum: pending | resolved"
  }
}
```

8.4 分支对象 (Branch)

```
{
  "id": "string (UUID)",
  "root_node": "string (node_id)",
  "leaf_nodes": "string[] (node_ids)",
  "assigned_agent": "string (agent_id)",
  "utility_function_id": "string",
  "temperature": "number [0, 1]",
  "status": "enum: active | merged | pruned"
}
```

---

9. 参考文献

1. Kuhn, L., Gal, Y., & Farquhar, S. (2023). Semantic Uncertainty: Linguistic Invariances for Uncertainty Estimation in Natural Language Generation. ICLR 2023.
2. Daulton, S., Balandat, M., & Bakshy, E. (2020). Differentiable Expected Hypervolume Improvement for Parallel Multi-Objective Bayesian Optimization. NeurIPS 2020.
3. Dung, P. M. (1995). On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. Artificial Intelligence, 77(2), 321-357.
4. Shapley, L. S. (1953). A value for n-person games. Contributions to the Theory of Games, 2(28), 307-317.
5. Anthropic. (2024). Model Context Protocol Specification. https://modelcontextprotocol.io/
6. Yuksekgonul, M., Bianchi, F., Boen, J., Liu, S., Huang, Z., Guestrin, C., & Zou, J. (2024). TextGrad: Automatic "Differentiation" via Text. arXiv:2406.07496.

---

10. 智能体提示词优化机制

10.1 概述

系统采用基于文本梯度的提示词优化机制，用于迭代改进分支管理智能体（BM）的系统提示词。该机制将提示词视为可优化参数，通过文本反馈进行更新。

核心原理：

- **变量**：智能体的系统提示词 P^{sys}
- **前向传播**：智能体基于 P^{sys} 和输入 I 生成输出 O
- **损失**：评估模型生成的文本批评 L^{text}
- **反向传播**：将批评逆向映射为提示词修改建议 ∇^{text}
- **更新**：$P^{'}_{sys} = \mathrm{TGD}(P^{sys}, \nabla^{text})$

10.2 优化约束

10.2.1 冻结策略

为避免多智能体同时更新导致的非平稳性问题，系统采用交替优化策略：

智能体	可优化性	理由
ACA（审计智能体）	冻结	作为系统的不变锚点，防止评估标准漂移
GA（调度智能体）	冻结	资源分配策略需保持稳定
RPA（需求智能体）	冻结	用户偏好解析需保持一致性

BM（分支管理智能体）	可优化	推理策略可根据任务反馈改进
ISA（信息智能体）	受限优化	仅优化查询策略，不优化验证逻辑
REC（编译智能体）	可优化	输出格式可根据用户反馈改进

10.2.2 信任区域约束

为防止提示词过度修正，每次更新必须满足：

- 保留原有核心指令
- 仅针对性微调失败案例相关的部分
- 修改幅度不超过原提示词长度的 20%

10.3 监督信号层级

系统采用加权复合损失函数：

$$\mathcal{L}^{total} = w^1 \cdot \mathcal{L}^{exec} + w^2 \cdot \mathcal{L}^{constraint} + w^3 \cdot \mathcal{L}^{gold} + w^4 \cdot \mathcal{L}^{critic}$$

10.3.1 硬性执行反馈 ($\mathcal{L}^{exec}$, $w^1 = 0.4$)

来源：代码执行结果、工具调用返回值、格式验证结果 **特性**：客观、不可篡改，作为系统的绝对锚点

10.3.2 约束合规性 ($\mathcal{L}^{constraint}$, $w^2 = 0.3$)

来源：ACA 智能体的合规性检查结果 **检查项**：

- 输出是否违反用户定义的 `ConstraintNode`
- 输出是否与已验证的 `FactNode` 矛盾
- 输出格式是否符合 JSON Schema

10.3.3 金标准对比 ($\mathcal{L}^{gold}$, $w^3 = 0.2$)

来源：预定义的高质量测试用例集 **用途**：防止模型退化，确保优化不破坏已有能力

10.3.4 模型批评 ($\mathcal{L}^{critic}$, $w^4 = 0.1$)

来源：其他智能体的文本评价 **约束**：仅在缺乏客观指标时使用，权重最低

10.4 批量优化

为降低文本梯度的随机性，系统采用批量梯度累积：

1. 收集 B 个任务实例的反馈（$B \geq 5$）

2. 聚合反馈为通用改进建议
3. 仅根据聚合建议更新一次提示词

聚合规则：

- 若 $\geq 80\%$ 的反馈指向同一问题，生成针对性修改
- 若反馈分散，不执行更新，标记为"需要更多数据"

10.5 归因路由

利用 Shapley 值进行责任分配，避免错误归因：

1. 分析完整推理轨迹
2. 估算每个智能体对最终结果的贡献度
3. 仅对贡献度超过阈值（$\phi > 0.3$）的智能体计算梯度
4. 贡献度低的智能体不参与本轮优化

10.6 版本控制与回滚

版本管理：

- 每次提示词更新创建新版本
- 保留最近 10 个版本的完整历史
- 记录每个版本的性能指标

回滚触发条件：

- 金标准测试集性能下降 > 5%
- 约束违规率上升 > 10%
- 用户显式请求回滚

```
{
  "prompt_version": {
    "id": "v1.2.3",
    "agent_id": "BM_001",
    "content": "...",
    "created_at": "2025-01-15T10:30:00Z",
    "metrics": {
      "gold_accuracy": 0.85,
      "constraint_compliance": 0.98,
      "avg_utility": 0.72
    },
    "parent_version": "v1.2.2"
```

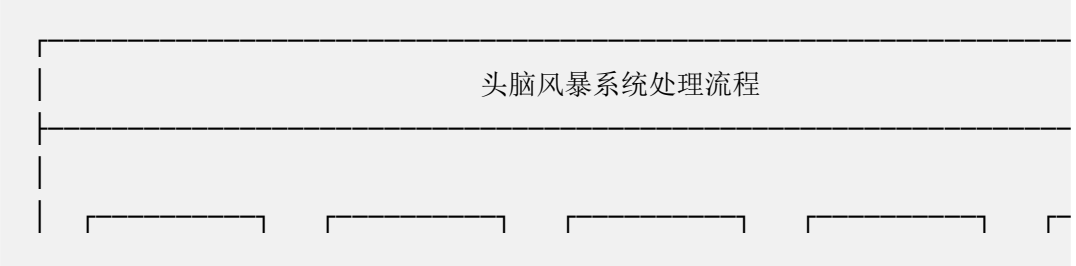
```
    }
}
```

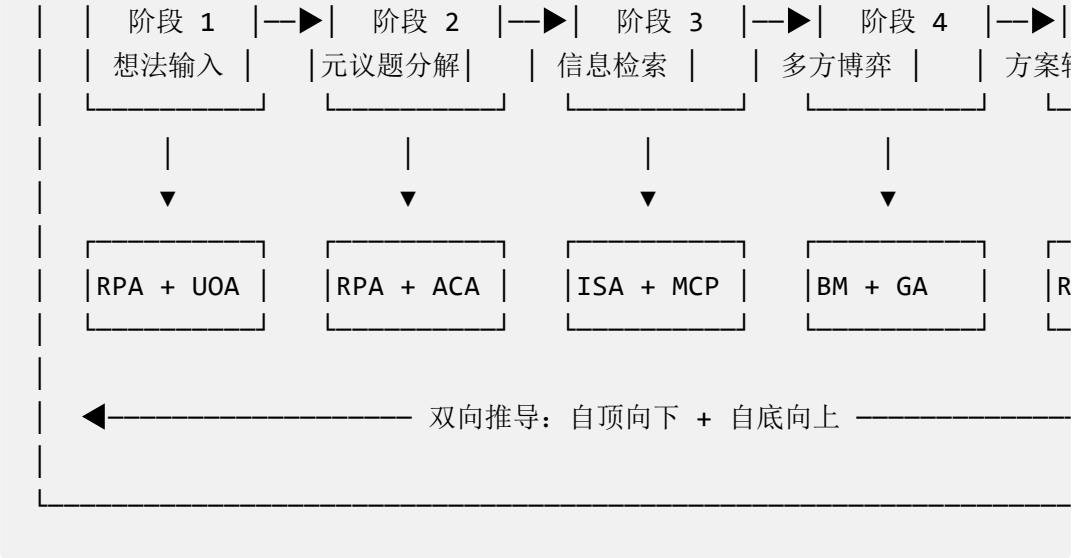
附录 A：术语表

术语	定义
超图 (Hypergraph)	图的推广，其中边（超边）可连接任意数量的节点
语义熵 (Semantic Entropy)	基于语义聚类的不确定性度量
Shapley 值	合作博弈论中衡量参与者边际贡献的公平分配方案
Nash 均衡	博弈论中没有参与者可以通过单方面改变策略获益的状态
Pareto 前沿	多目标优化中不存在支配关系的解集合
qEHVI	多目标贝叶斯优化中的采集函数，衡量期望超体积改进
MCP	Model Context Protocol，AI 应用与外部系统的标准化连接协议
NLI	Natural Language Inference，自然语言推理任务
LSH	Locality Sensitive Hashing，用于近似最近邻搜索的哈希技术
TextGrad	基于文本的自动微分框架，用于优化 LLM 系统中的文本参数
TGD	Textual Gradient Descent，文本梯度下降优化器
信任区域	优化过程中限制参数更新幅度的约束机制

附录 B：系统使用流程

B.1 完整处理流程





B.2 阶段详解

阶段 1：想法输入与偏好引出（Idea Input & Preference Elicitation）

持续时间：2-10 分钟 **参与智能体**：RPA, UOA **用户角色**：顶层管理者

步骤：

1. 用户提交初始想法或需求（自然语言）
2. 用户设定最终目标（期望达成的结果）
3. RPA 识别输入中的隐含约束
4. UOA 触发贝叶斯偏好引出：
    - 询问评估维度的相对重要性
    - 询问风险容忍度
    - 询问资源约束
5. 生成初始效用函数
6. 创建 `GoalNode`（初始想法）和 `ConstraintNode`（约束条件）

输出：

- 初始想法节点 G^0
- 最终目标节点 G^T
- 约束集合 C
- 效用函数 $u(x)$

阶段 2：元议题分解（Atomic Topic Decomposition）

持续时间：5-15 分钟 **参与智能体**：RPA, ACA **用户角色**：可选参与（同步模式）

步骤：

1. RPA 对初始想法执行递归分解
2. 验证每个子议题的原子性
3. 计算每个元议题的语义权重
4. ACA 检查分解的完备性和一致性
5. 同时对最终目标执行反向分解（识别必要条件）
6. 创建 `AtomicTopicNode` 和 `PendingNode`

输出：

- 元议题集合 $\{s^1, s^2, ..., s^n\}$
- 每个元议题的权重 $w(s^i)$
- 待验证节点集合（来自目标分解）

阶段 3：大规模信息检索（Large-Scale Information Retrieval）

持续时间：10-60 分钟（异步模式可更长） **参与智能体**：ISA, MCP Servers **用户角色**：顶层管理者（审阅检索结果）

步骤：

1. ISA 根据元议题权重排序检索优先级
2. 通过 MCP 协议连接外部数据源：
    - Web Search API：获取最新信息
    - 学术数据库：获取研究文献
    - 行业数据库：获取市场数据
    - 向量数据库：获取相似案例
3. 对每个检索结果执行语义熵验证
4. 高不确定性结果触发深度检索
5. 创建 `FactNode`，附加来源和置信度

输出：

- 事实节点集合 $\{f^1, f^2, ..., f^m\}$
- 每个事实的置信度 $S^{conf}(f^i)$
- 信息缺口报告（无法检索到的信息）

阶段 4：多方博弈与推理（N-Player Game & Reasoning）

持续时间：15-60 分钟 **参与智能体**：BM, GA, ACA **用户角色**：圆桌参会者（同步模式）或顶层管理者（异步模式）

步骤：

1. GA 初始化多个分支，每个分支代表一个独立观点
2. 为每个分支分配 BM，绑定异构效用函数

3. BM 执行双向推理：
    - 自底向上：从 `FactNode` 推导 `ClaimNode`
    - 自顶向下：从 `GoalNode` 分解 `PendingNode`
4. 双向匹配：`PendingNode` 与 `ClaimNode` 匹配形成闭环
5. BM 之间执行苏格拉底式诘问
6. BM 之间执行攻击/支持动作
7. ACA 验证攻击有效性
8. GA 计算 Nash 均衡
9. 若存在冲突，触发黑格尔辩证综合
10. 执行剪枝，移除低效用分支

输出：

- 收敛后的推理图 G^*
- 最优分支集合 B^*
- 综合节点（若有冲突）

阶段 5：可执行方案输出（Executable Plan Output）

持续时间：5-15 分钟 **参与智能体**：REC, UOA **用户角色**：顶层管理者

步骤：

1. REC 对因果视图执行拓扑排序
2. 询问用户当前条件：
    - 可用资源
    - 时间约束
    - 技能储备
3. 为每个 `ClaimNode` 生成具体行动步骤
4. 使用与元议题分解相同的逻辑拆分执行任务
5. 生成结构化输出文档
6. 附加置信度、证据链、风险提示

输出：

- 可执行行动计划（Markdown/JSON）
- 推理轨迹报告
- 置信度评估
- 风险提示和备选方案

附录 C：使用场景

场景 1：技术架构决策

问题描述： 企业需要选择微服务架构方案，涉及多个技术栈选项和业务约束。

输入示例：

```
需求：设计一个支持 10 万并发用户的电商平台后端架构。
约束：
- 预算不超过 50 万/年
- 团队熟悉 Java 和 Go
- 必须支持多数据中心部署
- 响应时间 < 200ms
```

处理流程：

阶段	操作	输出
输入处理	RPA 分解需求为 4 个约束节点	`ConstraintNode` × 4
偏好引出	UOA 询问："成本与性能哪个优先？"	效用函数权重
推理展开	BM 生成 3 个候选方案分支	分支 A: K8s + Java, 分支 B: Serverless, 分支 C: Go + gRPC
事实获取	ISA 查询云服务定价、性能基准	`FactNode` × 12
博弈收敛	分支 B 因成本超标被攻击	分支 B 被剪枝
综合生成	分支 A 和 C 的优势合并	综合方案：Go 微服务 + K8s
输出生成	REC 生成架构文档	技术方案 + 实施计划

输出示例：

```
{
  "recommendation": {
    "architecture": "Go 微服务 + Kubernetes",
    "confidence": 0.87,
    "estimated_cost": 420000,
    "key_components": [
      "API Gateway: Kong",
```

```
      "Service Mesh: Istio",
      "Database: TiDB (分布式)",
      "Cache: Redis Cluster"
    ]
  },
  "reasoning_trace": [
    {
      "claim": "Go 的并发性能优于 Java",
      "evidence": "mcp://benchmark/go-vs-java-2024",
      "confidence": 0.92
    }
  ]
}
```

场景 2：投资组合优化

问题描述： 投资者需要在多个资产类别中分配资金，平衡风险与收益。

输入示例：

```
需求：构建一个年化收益 8%+ 的投资组合
约束：
- 总资金 100 万
- 最大回撤 < 15%
- 股票占比 < 60%
- 必须包含债券
```

处理流程：

阶段	操作	输出
输入处理	RPA 识别 4 个硬约束	`ConstraintNode` × 4
偏好引出	UOA 询问风险偏好	风险厌恶系数 $\gamma = 2.5$
推理展开	BM 生成多个配置方案	分支 A: 保守型, 分支 B: 平衡型, 分支 C: 激进型
事实获取	ISA 获取历史收益率、波动率	`FactNode` × 20

博弈收敛	分支 C 因回撤超标被攻击	分支 C 被剪枝
输出生成	REC 生成配置建议	资产配置方案

输出示例：


```
{
  "portfolio": {
    "allocation": {
      "stocks": {"weight": 0.45, "detail": "沪深300 ETF 30%, 纳斯达
      "bonds": {"weight": 0.35, "detail": "国债 20%, 企业债 15%"},
      "alternatives": {"weight": 0.20, "detail": "黄金 ETF 10%, RE
    },
    "expected_return": 0.082,
    "expected_volatility": 0.11,
    "max_drawdown": 0.13,
    "confidence": 0.79
  }
}
```

场景 3：产品需求优先级排序

问题描述： 产品经理需要在有限资源下确定下一季度的功能开发优先级。

输入示例：

```
待排序功能：
1. 用户登录优化
2. 支付流程重构
3. 推荐算法升级
4. 移动端适配
5. 数据分析仪表盘

约束：
- 开发资源：3 人 × 3 个月
- 必须完成支付相关的合规要求
- 用户留存率是核心 KPI
```

处理流程：

阶段	操作	输出
输入处理	RPA 为每个功能创建 `ClaimNode`	`ClaimNode` × 5
偏好引出	UOA 询问各指标权重	留存率 0.4, 合规 0.3, 收入 0.3
推理展开	BM 评估每个功能的影响	影响评估矩阵
事实获取	ISA 获取历史数据、行业基准	`FactNode` × 8
博弈收敛	功能间的资源竞争建模为博弈	最优组合
输出生成	REC 生成优先级列表	排序结果 + 理由

输出示例：

```
{
  "priority_ranking": [
    {"rank": 1, "feature": "支付流程重构", "score": 0.91, "reason"
    {"rank": 2, "feature": "推荐算法升级", "score": 0.78, "reason"
    {"rank": 3, "feature": "用户登录优化", "score": 0.65, "reason"
    {"rank": 4, "feature": "移动端适配", "score": 0.52, "reason":
    {"rank": 5, "feature": "数据分析仪表盘", "score": 0.41, "reasor
  ],
  "resource_allocation": {
    "支付流程重构": "2人 × 2个月",
    "推荐算法升级": "2人 × 1.5个月",
    "用户登录优化": "1人 × 1个月"
  }
}
```

场景 4：法律合规审查

问题描述： 企业需要评估新业务模式是否符合数据保护法规。

输入示例：

```
业务描述：
- 收集用户行为数据用于个性化推荐
- 数据存储在海外服务器
- 与第三方广告平台共享匿名化数据
```

```
需要评估的法规：
- GDPR（欧盟）
- 个人信息保护法（中国）
- CCPA（加州）
```

处理流程：

阶段	操作	输出
输入处理	RPA 分解业务为 3 个数据处理活动	`ClaimNode` × 3
事实获取	ISA 检索法规条文	`FactNode` × 15
推理展开	BM 逐条评估合规性	合规评估矩阵
博弈收敛	不同法规要求的冲突识别	冲突超边 × 2
综合生成	生成合规改进建议	综合方案
输出生成	REC 生成合规报告	风险评估 + 整改建议

输出示例：

```
{
  "compliance_assessment": {
    "overall_risk": "medium",
    "by_regulation": {
      "GDPR": {"status": "non_compliant", "issues": ["跨境传输未获
      "PIPL": {"status": "partial", "issues": ["需要安全评估"]},
      "CCPA": {"status": "compliant", "issues": []}
    }
  },
  "remediation_plan": [
    {"priority": 1, "action": "实施标准合同条款(SCC)", "deadline":
    {"priority": 2, "action": "部署数据主体请求处理系统", "deadline"
    {"priority": 3, "action": "完成跨境数据传输安全评估", "deadline"
  ]
}
```

场景 5：研究方向选择

问题描述： 研究团队需要在多个潜在研究方向中选择下一阶段的重点。

输入示例：

候选方向：
A. 大模型推理效率优化
B. 多模态学习
C. 强化学习在机器人中的应用
D. 联邦学习隐私保护

评估维度：
- 学术影响力潜力
- 工业应用前景
- 团队技术储备
- 资源需求

处理流程：

阶段	操作	输出
输入处理	RPA 创建 4 个候选方向节点	`ClaimNode` × 4
偏好引出	UOA 询问各维度权重	学术 0.3, 工业 0.3, 储备 0.25, 资源 0.15
事实获取	ISA 检索论文引用、专利、招聘数据	`FactNode` × 24
推理展开	BM 为每个方向构建论证链	4 个分支
博弈收敛	方向间的资源竞争博弈	最优选择
输出生成	REC 生成研究规划	方向选择 + 路线图

输出示例：

```
{
  "recommendation": {
    "primary": "A. 大模型推理效率优化",
    "score": 0.84,
    "rationale": [
      "学术热度持续上升（2024 年论文增长 45%）",
      "工业需求强烈（推理成本是部署瓶颈）",
      "团队有 Transformer 优化经验"
```

```
    ]
  },
  "secondary": "B. 多模态学习",
  "roadmap": {
    "Q1": "文献调研 + 基准复现",
    "Q2": "方法设计 + 实验",
    "Q3": "论文撰写 + 投稿",
    "Q4": "开源 + 工业合作"
  }
}
```

附录 D：API 接口规范

D.1 会话初始化

```
POST /api/v1/sessions
Content-Type: application/json

{
  "mode": "sync | async",
  "config": {
    "max_iterations": 100,
    "token_budget": 50000,
    "temperature_schedule": "exponential"
  }
}

Response:
{
  "session_id": "uuid",
  "status": "initialized"
}
```

D.2 提交需求

```
POST /api/v1/sessions/{session_id}/requirements
Content-Type: application/json

{
```

```
  "content": "自然语言需求描述",
  "constraints": [
    {"type": "hard", "content": "约束 1"},
    {"type": "soft", "content": "约束 2", "weight": 0.8}
  ]
}

Response:
{
  "goal_node_id": "uuid",
  "constraint_node_ids": ["uuid1", "uuid2"],
  "status": "processing"
}
```

D.3 查询状态

```
GET /api/v1/sessions/{session_id}/status

Response:
{
  "phase": "reasoning | convergence | output",
  "progress": 0.65,
  "active_branches": 3,
  "pending_decisions": [...]
}
```

D.4 获取结果

```
GET /api/v1/sessions/{session_id}/result

Response:
{
  "status": "completed",
  "output": {...},
  "reasoning_trace": [...],
  "confidence": 0.85
}
```