Mutli Agent & Prompt engineering

hy

June 24, 2025

Abstract

1 Mutli Agent

1.1 基本定义

多智能体系统（Multi-Agent Systems, MAS）指由多个相互作用的、自治的（或半自治的）智能体组成的计算系统。这些智能体存在于**共享环境**中，通过感知环境、与其他智能体通信、进行推理和决策、执行动作来追求各自或共同的目标。

- **多：**系统包含不止一个智能体
- **智能体：**在环境中能够感知并行动的实体（物理的或软件的）
- **系统：**智能体及其交互构成的整体，表现出单个智能体无法实现的复杂行为

视图

$$\text{复杂问题} \xrightarrow{\text{分解}} \sum_{i=1}^{n} \text{相对简单的智能体} \xrightarrow{\text{交互}} \text{系统级解决方案} \tag{1}$$

1.2 架构

1.2.1 集中式架构

集中式架构采用星型拓扑结构，由中心控制器（协调者）作为系统核心枢纽。所有智能体仅与中心节点直接通信，中心控制器负责全局状态感知、任务分配、冲突消解。

(例如：在物流调度系统中典型实现为：AGV 小车实时上传位置信息至中央服务器，服务器基于 Dijkstra 算法计算最优路径后下发指令。该架构通信复杂度 $C(n) = O(n)$ 的优势体现在小型无人机编队（n 50）控制场景中，中心节点 1ms 级响应确保实时避障。但存在致命缺陷：当中心节点故障时系统全面崩溃（单点故障率 $P_f > 0.7$），且扩展至 200+ 智能体时通信带宽需求呈 $B = kn^2$ 增长，导致亚马逊 Kiva 仓库系统曾因中心服务器过载引发 12 小时瘫痪。当前主要应用于工业 PLC 控制、小型仓储机器人等封闭环境。)

1.2.2 分布式架构

分布式架构构建于 P2P 网络拓扑，各智能体具备完全自治性和局部决策能力。其核心机制包括：基于 Gossip 协议的状态传播、利用拍卖算法的任务分配以及采用梯度协商的冲突解决

（例如：在无人机集群搜索场景中，每架无人机通过 ZigBee 网状网络共享 3km 半径内环境数据，基于改进 Boids 模型实现自发编队。该架构通信复杂度 $C(n) = O(n^2)$ 导致大规模系统效率瓶颈，100 节点系统通信开销达集中式的 17 倍。但其卓越的鲁棒性（节点故障率 $P_f < 0.05$ 不影响系统功能）使其成为军事无人机群、区块链共识网络的首选，尤其适用于 GPS 拒止环境下的协同作战。）

1.2.3 联邦式架构

联邦式架构采用层次化混合设计，形成“联邦-成员”双层结构。系统划分为若干自治联邦，各联邦内采用集中/分布式混合管控，联邦间通过网关进行元协商。核心运行机制包含：联邦内基于合同网的任务分解、跨联邦的基于 Shapley 值的资源分配以及利用联邦学习的知识共享。

（例如：在智慧城市交通系统中，区域控制中心（联邦）管理本辖区信号灯集群，跨区域拥堵时各联邦中心通过 V2X 交换流量矩阵（$Flow_{ij} \in R^{n \times n}$），基于 Nash 均衡优化全局绿波带。该架构平衡了效率与鲁棒性，200 节点系统通信开销仅为纯分布式的 38%，同时支持动态联邦重组（重组延迟 $\tau < 500ms$）。典型应用于跨企业供应链协同、云边协同计算等异构系统集成场景。）

1.3 核心挑战

1. **协调问题**
    - 资源冲突
    - 解决方案

1.4 应用领域

领域	应用案例	代表技术
智能制造	柔性生产线	合同网协议
智慧交通	自动驾驶车队	V2X 通信
分布式能源	智能电网	博弈论调度
数字金融	算法交易	多智能体 RL
元宇宙	虚拟社会模拟	生成式代理

1.5 研究趋势

- **大规模 MAS**：→ 分布式学习算法优化
- **人机混合 MAS**：→ 人类行为建模
- **具身智能体**：→ 物理-虚拟空间协同
- **伦理与治理**：→ 去中心化决策机制

1.6 结论

多智能体系统通过**分布式智能**解决复杂问题，核心公式可概括为：

$$\text{MAS 效能} = \underbrace{\sum \text{个体能力}}_{\text{智能体能力}} + \underbrace{f(\text{交互结构})}_{\text{协同增益}} - \underbrace{g(\text{协调成本})}_{\text{系统损耗}}$$

2 Prompt Enginnering

2.1 Prompt 的定义与本质

在人工智能领域，Prompt（提示词）是用户与 LLM 交互的核心媒介。其本质是结构化文本指令，通过精准的语义表达引导模型生成预期输出。Prompt 不仅是简单的查询语句，更是包含任务目标、上下文约束和输出规范的完整操作指南。从技术视角看，Prompt 可形式化为 $Output = \mathcal{M}(Prompt|\theta)$，其中 $\mathcal{M}$ 代表语言模型，θ 为模型参数。有效的 Prompt 设计能显著提升模型输出的准确性、相关性和可控性。

2.2 Prompt 结构化设计 (CRISPE+CR)

1. **Capacity**：能力定位
2. **Role**：角色设定
3. **Insight**：背景洞察
4. **Statement**：任务陈述
5. **Personality**：输出个性
6. **Experiment**：实验要求

另外我根据自己的理解又加了两条（不用时填“无”即可）

1. **Context**：更多信息
2. **Requirement**：额外要求

这种分层结构使复杂任务可分解执行，同时降低模型的理解偏差。例如在 requirement 中，通过明确禁止使用特定库（如” 禁用 Pandas”）可强制模型采用基础算法实现。

2.3 高阶 Prompt 算法与技巧

Few-shot Prompting：用示例激活模型泛化能力，在 Prompt 中嵌入多个“输入-输出”示例，提升准确率。

Chain-of-thought Prompting：分步推理破解复杂问题，例如：“请写出详细过程”

Self-consistency：多次生成答案，取众数或置信度最高的结果，降低偶发错误。用投票机制增加结果鲁棒性。

自动学习 prompt

2.4 Multi-Agent 系统中的 Prompt Engineering

在 Multi-Agent 系统中，Prompt 升级为协调智能体协作的架构蓝图：

2.4.1 角色分工机制

每个智能体通过专用 Prompt 确定位置：例如

- *Coder-Agent*：专注于代码生成，Prompt 强调算法效率和语法规范
- *Validator-Agent*：负责质量检查，Prompt 内置漏洞检测规则
- *Analysis-Agent*：分析代码，Prompt 注重代码复杂度与规范性

这种分工使系统具备多领域协同解决问题的能力，如软件开发中同步处理编码、测试和文档化。

2.4.2 工作流

系统级 Prompt 构建工作流引擎：

1. 任务分解：将用户需求拆解为原子任务
2. 智能体调度：根据任务类型激活对应智能体
3. 质量闭环：设置评审迭代机制（如测试覆盖率 <80% 时自动重启编码流程）
4. 冲突仲裁：当智能体意见分歧时启动多数表决协议

这种转变使 Multi-Agent 系统能处理软件开发、科研分析等需要多阶段决策的复杂场景，其中 Prompt 本质上已成为定义智能体社会协作规则的” 宪法”。