# Why Do Multi-Agent LLM Systems Fail?

## 多智能体系统为什么会失败?

来自加州大学伯克利分校等机构的研究论文 *《Why Do Multi-Agent LLM Systems Fail?》* 首次对 MAS 的失败模式进行了系统性的、基于经验数据的分类研究。提出了第一个专门用于理解和诊断MAS失败原因的框架，他们将其命名为 **MAST (Multi-Agent System Failure Taxonomy，多智能体系统失败分类法)**。

### MAST 失败分类法

将超过200个任务中观察到的失败模式归纳为三大类共14种具体的失败模式。

通过分析7个流行的MAS框架（如MetaGPT, ChatDev等）在超过200个任务中的执行记录，动用了6名专家进行人工标注，最终总结出了一个包含3大类、14个具体失败模式的分类体系。

- **第一类：规约问题**
  - 这类失败源于**系统设计**层面的缺陷，比如系统架构、对话管理、任务或角色的定义不明确。
  - **根源：** 系统设计、角色定义和流程管理的缺陷。
  - **1.1 不遵守任务规约**：智能体未能遵循任务的关键约束或要求。
  - **1.2 不遵守角色规约**：智能体的行为超出了其被设定的角色范围，例如一个“代码编写者”去做了“测试者”的工作。
  - **1.3 步骤重复**：不必要地重复已经完成的步骤。
  - **1.4 对话历史丢失**：忘记了之前的对话上下文，导致行为脱节。
  - **1.5 不知道终止条件**：任务已经完成，但系统没有意识到，仍在继续运行。
- **第二类：智能体间未对齐**
  - **根源：** 智能体之间的沟通、协作和信息同步障碍。
  - **2.1 对话重置**：意外地重启对话，丢失了过程。
  - **2.2 未能请求澄清**：在遇到模糊指令时，没有主动询问，而是基于错误的假设继续执行。
  - **2.3 任务偏离**：在执行过程中逐渐偏离了最初的目标。
  - **2.4 信息隐瞒**：一个智能体获得了关键信息（比如正确的API用法），但没有分享给其他需要的智能体。
  - **2.5 忽略其他智能体的输入**：直接无视或未充分考虑其他智能体的建议或产出。
  - **2.6 推理与行动不匹配**：智能体的思考过程是正确的，但最终执行的动作却是错误的。

- **第三类：任务验证失败**
  - 这类失败源于系统对最终产出的**质量控制**不足。
  - **根源：** 缺乏有效、深入的结果验证机制。
  - **3.1 过早终止**：在任务所有目标达成前就提前结束。
  - **3.2 未验证或验证不完整**：没有检查或只做了非常表面的检查（例如，只检查代码能否编译，不检查逻辑是否正确）。
  - **3.3 错误验证**：进行了验证，但未能发现其中明显的错误。

论文得出一个关键结论：**MAS 的失败不仅是底层 LLM 模型能力（如指令遵循）的局限，更深层次的原因在于系统设计和组织协调的失败。** 就像一个管理混乱的团队，即使成员再优秀，也难以高效协作。因此，简单的提示工程优化是治标不治本的，必须进行更根本的系统架构和交互流程的重新设计。

## 如何诊断自己的多智能体系统是在哪一步失败了？

开发了一个自动化诊断流程LLM-as-a-Judge，以下是该流程基于OpenAI o1模型的表现情况：

Table 2: Performance of LLM-as-a-judge pipeline

| Model | Accuracy | Recall | Precision | F1 | Cohen's $\kappa$ |
|---|---|---|---|---|---|
| o1 | 0.89 | 0.62 | 0.68 | 0.64 | 0.58 |
| o1 (few shot) | 0.94 | 0.77 | 0.833 | 0.80 | 0.77 |

## 改进 MAS 的方法和策略

1. 战术性方法
   - **改进Prompt以明确角色和任务**：
     - 为每个智能体编写更详细、更明确的系统提示词。清晰地界定其职责、权限和禁止的行为。
   - **增加自我验证/反思步骤**：
     - 在Prompt中要求智能体在完成任务后，增加一个“反思”步骤，回顾自己的推理过程和最终结果是否符合初始要求。
   - **优化智能体交互模式**：
     - 设计更有效的对话流程。例如，可以让多个智能体独立提出解决方案，然后进行“同行评审”或辩论，而不是简单地线性传递任务。
2. 结构性策略
   - **强化验证机制**：

- 这是论文**最强调**的一点。不能依赖于单一、表面的验证智能体。需要建立多层次的验证体系，例如引入专门生成“单元测试”的智能体，以及进行“端到端验收测试”的智能体。

- **设计标准化通信协议**：
  - 减少智能体之间模糊的自然语言交流，转而使用结构化的格式（如JSON）进行通信，明确定义意图和参数。
- **引入外部记忆和状态管理**：
  - 为系统增加一个共享的、持久化的记忆模块（例如向量数据库），以解决长上下文丢失问题，并确保所有智能体都基于一致的信息进行决策。

下图是优化前和优化后（改进Prompt和强化验证机制两种改进措施）对比效果：

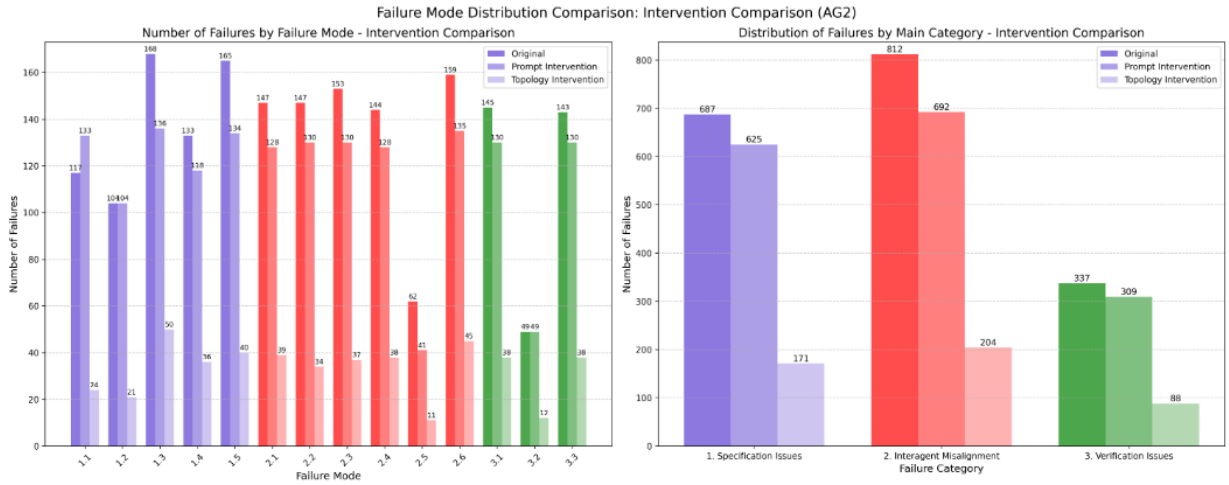


Figure 8: Effect of prompt and topology interventions on AG2 as captured by **MAST** using the automated LLM-as-a-Judge

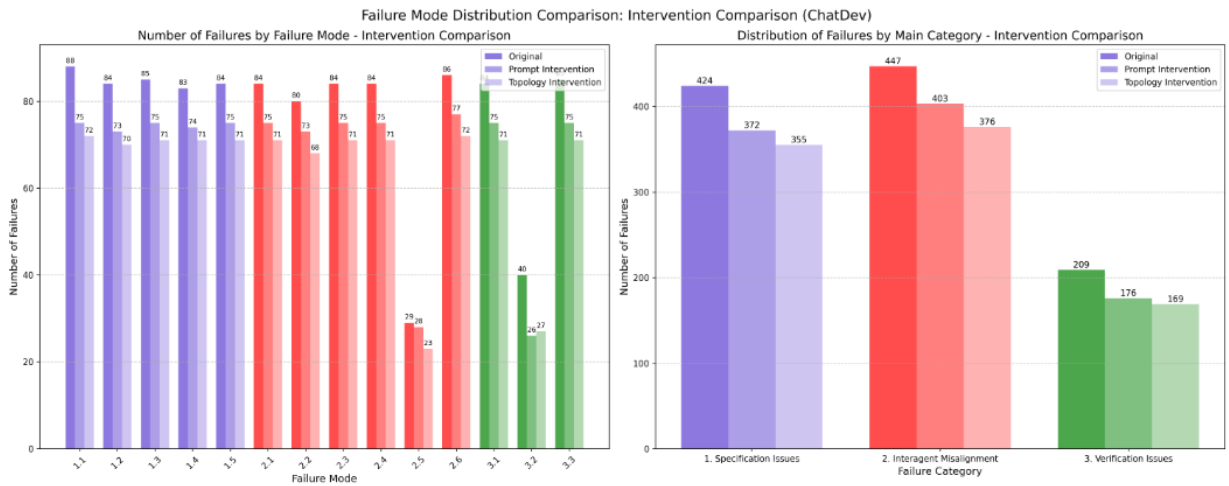


Figure 9: Effect of prompt and topology interventions on ChatDev as captured by **MAST** using the automated LLM-as-a-Judge

## 总结

最根本的原因在于智能体交接出现了问题，如果想构建一个好的多智能体系统，有两种思路，要么就是管理好智能体之间的沟通协作，这种对上下文工程要求很高；要么就是改变思路用一种最简单的单线程“多智能体”架构，即只保留一个主线程，对于简单的问题就用主智能体完成，对于复杂问题，会生成一个自身的“克隆”作为子智能体，通过最多一个分支来保证问题质量。这种设计允许主智能

体和子智能体之间进行协作，但它们不像是传统意义上的多智能体系统中各个智能体之间的平等互动和自主决策。更像是层级关系。比如claude code就是这种架构。

## 思考：我们应该致力于修复它，还是彻底改变思路？

在2025年的技术水平下，多智能体间的协作由于上下文共享不足和决策分散，系统极其脆弱。与其投入巨大精力解决难以捉摸的跨智能体通信问题，不如回归到**构建一个拥有强大上下文管理能力的、更强大的单智能体**。

参考资料：

- https://papers-pdfs.assets.alphaxiv.org/2503.13657v2.pdf