

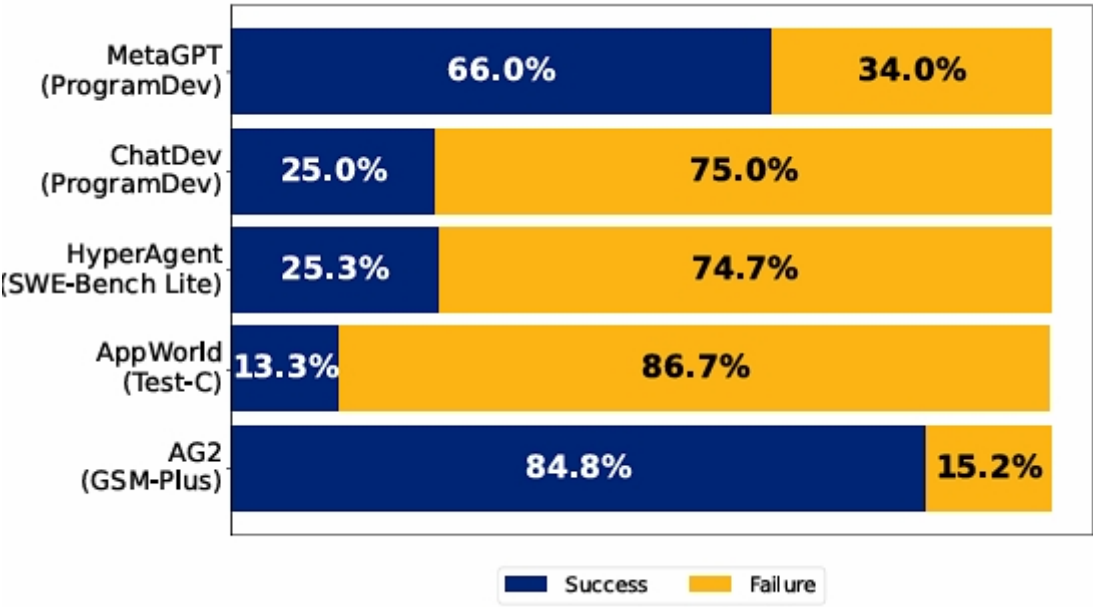
# 那些年失败的多智能体

🔧 有团队合作经验的各位，可能都经历过合作半天还不如不合作的经历。例如，领导、产品和程序员各有各的想法，沟通鸡同鸭讲，做出来的产品能跑就行。这种现象正是多智能体系统的真实写照，论文《Why Do Multi-Agent LLM Systems Fail》分析了导致多智能体失败的原因，并提出了可能的解决方案。

## 背景介绍

**MAS (Multi-Agent Systems) 定义：**LLM-based 智能体定义为一个具有提示规范（初始状态）、对话记录（状态）以及与环境交互能力（动作）的实体。**MAS**由多个智能体组成，通过编排实现协作，以实现集体智能。

**研究背景：**MAS允许多个 LLM 智能体协作完成任务，理论上能比单个智能体取得更好的效果（将复杂任务拆解，每个Agent只完成其中一部分，理论上肯定比一个Agent全包好）。但**现实中MAS却在基准测试上的性能提升仍然微乎其微**，下图中主流的MAS框架ChatDev在某些任务中的正确率低至25%。



## 论文贡献：

1. 分析了五种流行的 MAS 框架，涉及 150 多个任务，每个轨迹平均超过15,000行文本。提出多智能体系统失败分类法（**Multi-Agent System Failure Taxonomy, MASFT**）将失败模式分为3类：
  - **角色规范和系统设计问题：**比如任务的定义不够清晰，角色定义不明确，或者系统流程设计本身存在缺陷。这个问题很普遍，不止出现在多agent场景。
    - 举例：分工不明确，导致不同agent之间有权责重叠。

- **智能体之间的不协调**：比如不同智能体之间的目标不一致，信息隐瞒，或者它们可能忽略关键信息。
    - 举例：一个团队写代码（包括leader、产品、程序员），沟通的时候大部分时间都在讨论不重要的东西，写出来的代码质量很低。
  - **任务验证和终止问题**：比如系统无法正确判断任务是否完成，或者验证不完整不正确。
    - 举例：大模型生成的内容可能会存在错误，这种错误不仅来自幻觉，也来自于搜索时无法过滤掉低质量网页和资料，这一问题在多智能体中被放大了，从而无法验证自己生成的内容是否正确，以及何时终止生成。
2. 为了验证分类的准确性，邀请三组专家对失败模式进行分析，**Cohen's Kappa** 得分为 0.88，证明分类结果非常可信。
  3. 此外，论文将**MAS**与“大语言模型作为裁判”（**LLM-as-a-Judge**）结合在一起，使用gpt-o1以支持高效的评估，通过与三个人类专家对10个轨迹的注释进行交叉验证，Cohen's Kappa 得分为 0.77。
  4. 失败模式需要更复杂的解决方案，单纯的清晰智能体角色或者优化智能体协作模式无法彻底解决以上问题。

不要把失败模式全甩锅给LLM自身的局限性，许多MAS失败源于智能体之间的交互挑战，而不是单个智能体的局限性。



### 谈谈我对多智能体的理解

- 人类受限于自身知识的和生产效率有限性，人类协作可以让每个人发挥自己的长处并且提高生产效率。
- 多智能体与人类协作不一样，大模型一直在往通用化方向发展，单纯按照任务类型去给多个agent分配任务（比如分别做前端、后端、算法），效果可能不如都交给一个agent做（多智能体对任务理解不同，会因为沟通彼此之间的思路、数据格式、具体代码等浪费很多时间，效率反而可能不如单agent）。
- MAS更适合于群体智能场景，也就是多agent做的事情基本一致，比如多个agent互相对抗（狼人杀），互相启发（共同审阅一篇论文）。

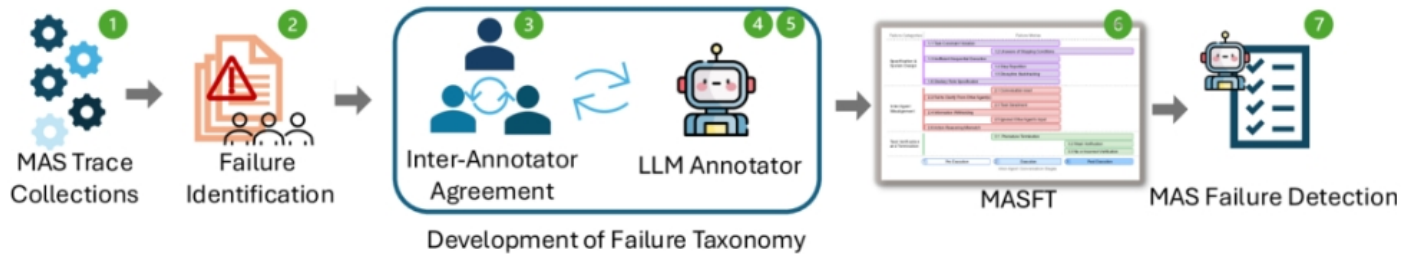
## 失败模式分类

这部分不做重点讨论，只简要介绍。

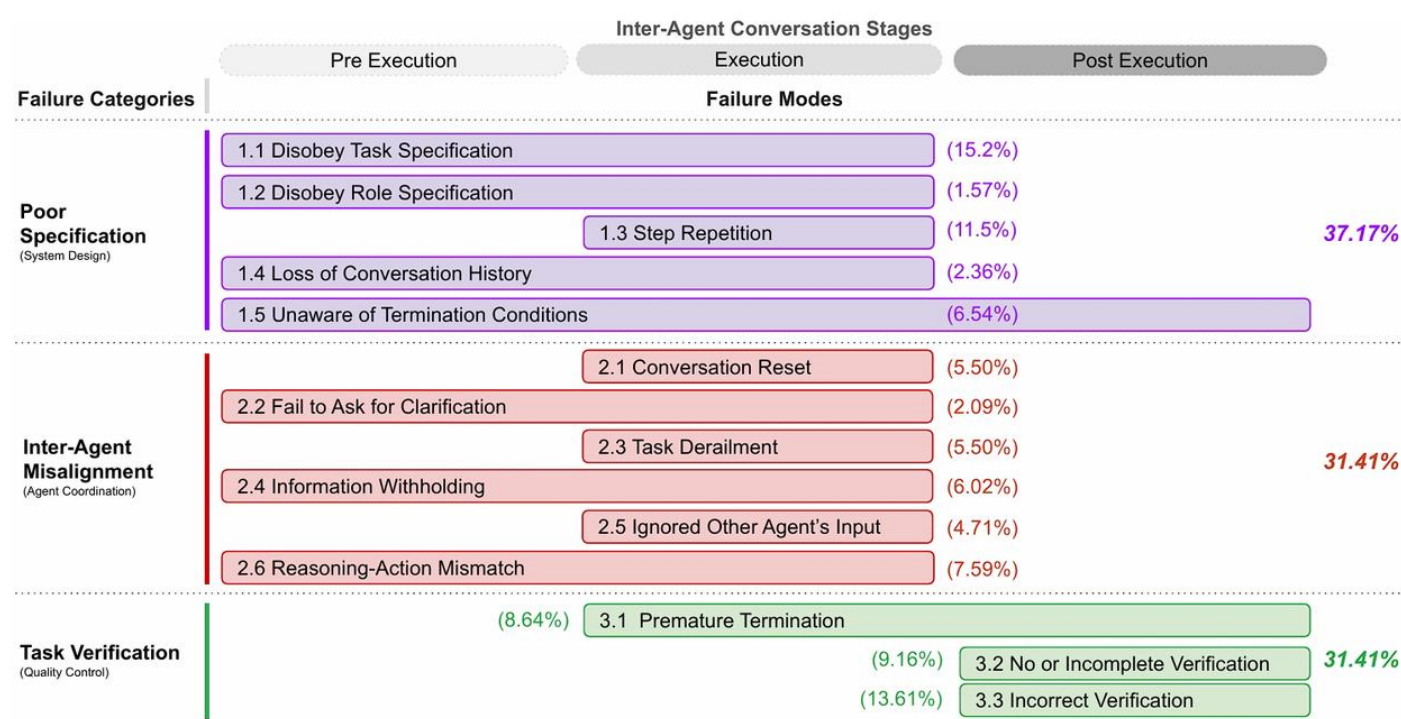
论文创建了一个结构化的分类体系MAST，该体系涵盖了多种失败模式，为每种模式提供了深入的见解，并且分类结果具高度的一致性。

- 采用**Grounded Theory**（GT）方法，通过理论抽样、开放式编码、持续比较分析、备忘录和理论化等步骤，迭代地收集和分析多智能体系统（MAS）的执行轨迹。

- 通过理论抽样确保所选MAS和任务的多样性，分析了HyperAgent, AppWorld, AG2, ChatDev, MetaGPT 5个MAS。
- 通过持续比较分析，注释者识别失败模式，并将其与现有代码进行比较，确保代码的准确性和一致性。直到达到理论饱和，即额外数据不再提供新见解。
- 进行一轮初步分类和两轮优化分类，Cohen's Kappa 达到0.84
- 为实现自动化注释，实现了LLM-as-a-Judge，few-shot形式gpt-o1的Cohen's Kappa 达到0.77。



失败模式分析



上图将MAS失败按照与发生阶段和失败原因进行分类，横轴表示发生阶段，可以分为**执行前**（任务分配、角色定义）、**执行中**（智能体交互、调用工具）和**执行后**（任务验证）。接下来重点介绍失败原因。

规范与系统设计问题

这类失败源于系统设计本身的缺陷、会话管理糟糕、用户任务指令的不明确、或者智能体未能遵循其角色和职责。

- Disobey task specification（违反任务、规范）**：未能遵守给定任务的指定约束或要求，导致次优或错误的结果。

- **例子：**开发国际象棋游戏时，输入应该遵循国际象棋记谱法（如'Ke8', 'Qd4'），而最终生成的游戏却要求输入棋子移动前后的坐标 (x1, y1), (x2, y2)。
- **Disobey role specification（违反角色规范）：** 未能遵守分配角色的既定责任和约束，智能体1越俎代庖做其他智能体2的活。
  - **例子：**在ChatDev的需求分析阶段，CPO（首席产品官）智能体有时会越权，承担CEO的角色，自行定义产品愿景并做出最终决策。
- **Step repetition（步骤重复）：** 不必要地重复先前已完成的步骤，可能导致任务完成的延迟或错误。
  - **例子：**HyperAgent中的“导航员”智能体为了实现Line3D类而反复尝试，即使已经实现了。
- **Loss of conversation history（会话历史丢失）：** 上下文意外截断，导致智能体忽略最近的交互历史并恢复到先前的对话状态。
  - **例子：**HyperAgent在解决一个编程bug时，一开始决定用scikit-learn模型替换所需的lightgbm库，但在后续交互中，它似乎忘记了这个决定，又回过头来尝试安装lightgbm。
- **Unaware of termination conditions（不知道终止条件）：** 智能体不知道何时应该结束交互，导致不必要的对话持续。
  - **例子：**在AG2解决一个数学问题时，即使已经给出了正确的答案，或者问题无法解决，智能体仍然反复要求继续进行。

## 智能体之间的不协调

这类失败发生在智能体之间的沟通和协作环节，存在无效沟通、行为冲突、偏离初始任务、互相误解等问题。

- **Conversation reset（会话重置）：** 对话意外或无端的重启，可能导致上下文和交互中取得的进展丢失。
    - **例子：**同Step repetition。
  - **Fail to ask for clarification（未能请求澄清）：** 面对不清楚或不完整的数据时没有请求更多信息，而是基于猜测行动，可能导致错误的操作。
    - **例子：**AppWorld 中的“主管”智能体指示“电话”智能体使用电子邮箱ID作为用户名。电话智能体在阅读文档后发现正确的用户名应为电话号码，但仍继续使用错误的邮箱ID，导致出现错误。
  - **Task derailment（任务偏离）：** 偏离给定任务的预期目标或重点，可能导致无关或无效的操作。
    - **例子：**AG2在解决一个数学问题时，可能中途被某个计算细节带偏，开始解决一个完全不同的问题，或者在找到正确答案后又继续进行不相关的计算。
  - **(Information withholding) 信息隐瞒：** 未分享智能体自身拥有的重要数据或见解。
    - **例子：**HyperAgent 的“导航员”有时找到了潜在解决方案，但没有将其完整传达给“planner”，导致后者无法做出正确决策。
-

- **Ignored other agent' s input（忽略其他智能体的输入）：** 忽视或未能充分考虑系统中其他智能体提供的输入或建议。
  - **例子：**在Multi-Agent Peer Review系统中，智能体1收到了智能体2对其数学解题过程的正确反馈，智能体1口头上承认了反馈，但没有发现其解决方案与其自身解决方案之间存在矛盾，在最终答案中仍然坚持自己最初的错误结果。
- **Reasoning-action mismatch（推理-行动不匹配）：** 智能体的逻辑推理过程与其实际行动之间存在差异。
  - **例子：**HyperAgent 的“导航员”已经发现了正确答案，但告诉“planner”是无关的建议。

## 任务验证与终止失败

这类失败发生在任务验证阶段，包括由于过早执行终止而导致的故障，以及缺乏保证交互、决策和结果的准确性、完整性和可靠性的机制。

- **Premature termination（过早终止）：** 在交换所有必要信息或实现目标之前结束对话、交互或任务。
  - **例子：**HyperAgent 的“编辑器”智能体声称已经完成了对代码的修改，但实际上并没有执行修改操作，却提前结束了自己的任务环节，导致后续依赖该修改的步骤失败。
- **No or incomplete verification（无或不完整验证）：** 系统缺少验证步骤，或者验证步骤未能覆盖所有关键方面，导致错误或不一致被遗漏。
  - **例子：**AG2 系统中把鱼的数量和它们的成本搞混了，但没有验证。
- **Incorrect verification（验证错误）：** 存在验证步骤，但验证本身是错误的或无效的，未能发现实际存在的问题。
  - **例子：**MetaGPT 在实现棋类游戏时，单元测试可能只覆盖了最基本的情况（如兵的移动），没有覆盖非兵棋子的复杂移动规则，却错误地认为验证通过。



### MAS失败模式的一些观察

- 这些失败模式分布相对均匀，没有单一错误类别主导失败发生。
- 失败模式可能具有连锁效应。
- 虽然验证是最后一道防线，但并非所有问题都由验证不足引起。
- MAS的失败模式违反了核电站、航空管制等高可靠性组织（HRO）的规则，如“不遵从角色规范” (FM-1.2) 违反了“极端层级分化”，“未能请求澄清” (FM-2.2) 违反了“尊重专业知识”。

## 设计更好的多智能体系统

作者将其分为战术性方法和结构性策略两类，前者针对特定失败模式进行的直接小修小补，后者涉及对整个系统结构进行修改，从基础架构层面提升 MAS 的鲁棒性和可靠性。




战术性方法

- **提示词优化**：明确提示词中的任务和每个智能体的角色，并鼓励智能体之间进行主动对话，如果存在不一致，智能体应能够重试任务。
- **自我验证**：通过重新陈述解决方案、检查条件并测试错误，智能体可以发现潜在问题。
- **模块化设计**：采用模块化方法，使用简单且定义明确的智能体，而不是复杂的多任务智能体。
- **交叉验证**：不同智能体可以提出多种解决方案，并通过交叉验证确保结果的准确性。

结构性策略

- **加强验证**：验证是抵抗模式失败的最后一道防线，不充分的验证机制是MAS失败的重要原因之一。
- **标准化通信协议**：基于大语言模型的智能体主要通过无结构的文本进行通信，这可能导致歧义。通过明确定义意图和参数，可以提高智能体之间的对齐度，并在互动过程中进行正式的连贯性检查。
- **强化学习**：奖励任务相关行为并惩罚低效行为。
- **概率置信度**：当智能体的置信度低于某个阈值时，它们可以暂停以获取更多信息，从而避免错误决策。
- **记忆和状态管理**：增强上下文理解并减少交流中的歧义。

附录：


 **Cohen's Kappa：一种衡量分类一致性的指标**

定义  $p_o$ ：评估者之间观察到的相对一致性， $p_e$ ：机会一致的假设概率

$$kappa = (p_o - p_e) / (1 - p_e)$$

计算示例：

用户1/ 用户2	Yes	No
Yes	25	10
No	15	20

 • 计算  $p_o$ ：

$$p_o = (\text{都说是} + \text{都说不是}) / (\text{总分})$$

$$p_o = (25 + 20) / (70) = 0.6429$$

- 计算  $p_e$  :

用户1说“是”的总次数除以响应总数\*用户2说“是”的总次数 + 用户1说“否”的总次数除以响应总数\*用户2说“否”的总次数

$$P(\text{“是”}) = ((25+10)/70) * ((25+15)/70) = 0.285714$$

$$P(\text{“否”}) = ((15+20)/70) * ((10+20)/70) = 0.214285$$

$$p_e = 0.285714 + 0.214285 = 0.5$$

- 计算Kappa, Kappa值越大说明一致性越强

$$kappa = (p_o - p_e) / (1 - p_e) = 0.2857$$