尽管大型语言模型（LLM）在多种自然语言处理任务中取得了显著的成功 [1]，但将其应用在多智能体系统中，处理需要深度、多步推理的复杂问题时，仍面临三个核心挑战： ①多智能体进行信息交互时，因为上下文长度限制和背景角色设定不同，会造成严重信息差；②博弈式互评时，容易受到批判方干扰，难以兼顾“坚持己见”与“博采众长”；③单智能体幻觉易引发全局误差，容易造成信息损失、增加或篡改[2]。

传统的 LLM 在进行长链条推理时， 其内部的记忆与知识调用机制如同一个“黑箱”，容易产生信息遗忘或事实扭曲（即幻觉）； 同时，其线性的、端到端的生成方式使得中间的思考步骤难以被评估与干预，导致最终结果的质量难以保证。针对幻觉问题,研究者们提出了多种控制策略，可概括为主动验证[3]、架构优化[4][5]和知识增强[6][7]。为量化评估幻觉控制效果，研究者们也提出了专门的关键绩效指标（KPIs）体系[5]，包括幻觉发生率、事实一致性得分、不确定性校准误差等[8]，通过第四级评估智能体对系统输出进行自动量化分析[5]。

近年来，多智能体框架与对抗性机制也成为有效应对幻觉的策略之一。例如，Sun等人[9] 引入了对抗与投票机制，用于检测并减少幻觉，突出了协作式验证过程的作用。类似地，Lin等人[10]设计了多智能体辩论框架，通过对抗式信息验证减少幻觉，在多步骤任务链中引入辩论机制后，事实一致性得分从 0.62 提升至 0.81，尤其在需要跨模态推理的任务中，幻觉发生率降低 55%。此外，Yoffe等人[11] 结合量化性能评分机制，以 “量化不确定性 + 融入辩论交互” 为核心，将 “隐性置信度” 转化为 “可利用的决策依据”，打破传统辩论仅依赖答案内容的局限,直接促进了幻觉的减少和可预测性的增强。

针对上述一系列挑战，我们也寻求在框架与机制上创新，设计并实现了 NagaAgent ，旨在提升推理过程的透明度、鲁棒性与最终方案质量的智能体系统。本研究经过一系列的数学模型推导，确定并设计了三大核心模块，构建了一套Agent的整体框架，有助于解决前文提出的三大挑战。

其中，核心模块 1 为交互图生成器，其定位是框架的初始化驱动模块，主要为后续所有流程搭建结构化协作基础，具体承担三大核心功能：

功能一是角色生成，会根据输入的论文研究主题或任务目标，自动生成与任务强相关的功能性角色。图中角色由大模型api自动生成，生成器称作Distributor，不应该自动枚举。有一个生成角色的system prompt并且格式化从中提取角色名称和角色职能描述，然后为后续的角色加上当前的角色分工协作图，并给出它们本身具有连接哪一些角色的权限。同时，还会构建角色之间的通信链路，会依据各角色的职责逻辑与协作关系，制定定向的信息传输规则，以游戏开发场景为例，以产品经理为起始节点，允许的传输路径包括 “产品经理⇄程序员” “产品经理⇄美工” “程序员⇄测试人员”等，距离比较远的角色之间如果需要通信，需要经过如产品经理等角色的中转，以防模型在未经评估的情况下盲目增加需求。二者需通过中间节点完成间接信息传输，以此避免跨角色职责的无效沟通。每个角色在完成自身当前阶段的任务后，可根据任务输出结果与下一阶段的需求，自主选择符合链路规则的目标角色传递信息。注意，我们全流程都是大模型生成的，严禁采取任何形式的硬编码。

功能二是角色提示词生成。每个角色都有各自的system prompt采用一个Prompt Generator进行专门的生成，每个角色名字和描述都进行一次生成。比如在游戏开发相关研究场景中，会生成 “产品经理”“程序员”“美工”“测试人员” 等角色，且每个角色都对应明确的任务职责与输出要求，确保角色功能与研究需求精准匹配。

功能三是数据表格输出，我们需要设计多种问题，然后模型对于这个问题生成的角色全部输出到xlsx表格，每一行分别对应一次询问，第一列是问题，第二列是生成的角色列表，以json形式存储。第三列是角色之间的连接关系，以[[“”,””],[“”,””]这样的形式存储，第四列是角色1prompt，第五列是角色2prompt等等，以此类推。

同时，我们增加了迭代轮次的约束，确保模型不会陷入思维盲区。每一个自博弈模块如果思索轮次过多就会强制进行交接，由交付给它任务的模块判断任务完成情况如何。需要注意，设计中没有枚举。

流程是用户发送给博弈模块一个问题，博弈模块就进行相应的推理，得到回应后传回给用户。首个节点默认设置为需求方，由需求方传递消息给第一个节点（如产品经等），产品经理觉得产品可以了就可以选择传给需求方，然后需求方接收到后后就返回给用户。这样可以避免突兀的返回选项，而是将提出这个需求的用户融入这个图中。注意，需求方不由distrributor生成，你只需要告诉distributor我们有这个逻辑并禁止它生成 需求方就行。需求方相当于是一个用户输入输出的形式节点，由程序默认生成。

核心模块 2 为模型自博弈模块，其定位是保障生成内容的质量与创新性，通过 “Actor-Criticizer-Checker” 三组件的协同工作，形成 “生成 - 批判 - 评估” 的闭环优化机制。组件占比为”1: n: 1”。其中，Actor 组件的核心职责是功能生成，它会接收上游交互图生成器输出的角色指令与任务目标，执行具体的模块功能，比如论文框架生成、实验方案设计、数据结果分析等，并输出初始的任务成果，为后续的批判与评估提供基础素材；Criticizer 组件的核心职责是成果批判与优化建议，它会基于任务目标与领域知识，例如研究的创新要求等，对 Actor 输出的初始成果进行多维度批判，精准识别其中存在的逻辑漏洞、创新性不足、细节缺失等问题，并针对性地提出优化建议，同时还会为 Actor 的成果表现进行打分（记为Critical score），为后续的评估提供参考依据，以及对自己上一次建议被采纳或反驳的满意度进行打分（记为 Satisfaction score）；Checker 组件的核心职责是创新性评估（该组件设计改进了的现有Philoss模块），它会以预设的评估维度为输入，返回研究思路的新颖性（记为Novel score），通过构建量化评估模型对当前思路或成果的创新性进行打分。philoss模块的原理是冻结模型其他参数，然后在模型内部置入一个MLP层，用这个层去预测模型内部的隐藏层State状态。当前是State1，MLP基于State1预测State2，是对下一个时刻的预测。如果预测和真实的State2误差大，就说明模型的思考模式很有趣，创新度大；如果模型的误差小，那就说明模型并没有从更新预测模式，文本比较无趣。Checker是这个模块的进一步改进实现。实现细节是，我们采用api的形式调度大模型，获取大模型的输出文本块。然后我们在本机上面开一个7b的qwen2.5-VL小模型，然后给小模型植入philoss模块。因为其他参数冻结了，所以我们可以原样复用小模型参数。我们将大模型的输出按照100个token一组切分为若干个文本块，然后philoss基于State x对State x+1的预测就基于这些文本块。小模型对下一个词的预测不再是基于自回归的，而是基于大模型输出的文本块，来更新自己的隐藏层状态Checker仅对回答的超出常规思维的程度进行打分，不判断正确性。该分数将作为判断后续是否进入下一轮优化流程的关键决策依据，确保每一轮优化都能切实提升研究的创新性与质量。在提示词的设计中，我们引入了 “思维向量” 和“思维栈” 的概念，旨在让模型能够以最清晰的视角去看到此前所有任务的更新迭代。其核心设计围绕 “锚定核心思考方向，保障任务一致性” 展开。一方面，在提示仓库输出初始提示词时，会自动为其附加 “思维向量” 约束，该约束会明确当前任务的核心目标。我们要求模型采用function-call的形式更新思维向量，同时模型在当前的思维向量下面可能还会存在更加细化的解决目标，就会让它以”<belief 角色 层级>”的形式进行存储。在每次提示词输入的末尾都会将当前的思维栈注入。例如 “</belief programmer 1>围绕 XX 领域的 XX 问题，设计具有 XX 创新点的解决方案</belief programmer 1>”。通过这些约束确保模型在不同思考阶段，无论是初步生成、批判优化还是细节补充阶段，均能始终围绕核心目标展开思考，有效避免思路发散或偏离；另一方面，该功能还支持 “思维向” 的动态调整与 “任务外包”，当研究过程中需要将部分子任务，例如数据预处理、文献综述初稿撰写等，外包给其他模块或角色时，可将 “思维向” 约束作为核心要求同步传递给承接子任务的模块或角色，确保外包任务的输出结果与整体任务目标保持一致。

核心模块 3 为群体共识记忆模块。我们基于群体共识理论，采用 Graph RAG 技术实现信息的结构化存储与高效检索 —— 将论文创作过程中产生的角色交互数据、批判优化记录、成果版本迭代等信息，以图结构形式构建关联图谱，既保留信息间的逻辑关联（如某轮优化建议与对应成果的关联、不同角色任务输出的衔接关系），又能在后续流程中快速定位历史信息，避免重复劳动（如无需重新生成已确定的角色链路、无需重复评估相同创新点），保障研究的连续性。