大模型智能体的评测标准与能力影响因素: 简要报告

赵梓霖
zhao_zilin@outlook.com

摘要(Abstract)——大语言模型智能体（LLM Agent）代表了从静态语言模型向自主决策系统的演进方向。本报告尝试梳理LLM Agent的评测标准与能力影响因素，希望为该领域的研究者与工程实践者提供一个参考框架。
在评测维度方面，本报告整理了双层评测视角：核心能力层关注规划推理、工具使用、记忆管理等智能体的基础能力，总结了各能力的实现方式、常见评测指标与典型问题；系统工程层讨论性能效率、任务完成度、鲁棒性以及安全对齐等实际应用中需要考虑的质量属性。这一分层方式试图兼顾理论探讨与工程实践的不同关注点。在影响因素方面，本报告归纳了模型内在因素（架构设计、训练方式、知识范围）与系统设计因素（提示工程、工具集成、多智能体协作、部署优化）可能存在的相互作用。现有研究表明，智能体能力并非仅由基础模型规模决定，而是模型能力与系统设计共同作用的结果。
本报告还提出三个尚待深入探讨的问题：多维度评测指标间如何权衡、能力可解释性不足如何影响评测有效性、以及长期运行中价值对齐可能发生的变化。这些问题反映了当前评测体系存在的一些局限，也是未来研究可以关注的方向。
通过整理ReAct、多智能体系统等典型架构的相关工作，本报告尝试呈现从单体工具调用到分布式协同的技术发展脉络，希望能为理解LLM Agent的特点与趋势提供一些思路。
关键词(Key words)——大模型智能体、能力评测、规划推理、多智能体系统、价值对齐

1 引言

1.1 LLM 到 LLM Agent 的演化背景

通用大语言模型（比如ChatGPT-3）最初主要作为静态接口，给定输入后生成单轮输出[1]。随着实际应用场景扩展，有三类需求逐渐超出"纯 LLM"范式（一问一答的文本自回归范式）的承载能力[2]。

第一类是工具使用需求。许多场景需要调用搜索、数据库、代码执行器、企业内部 API 等外部工具。用户期望模型不仅能"回答"问题，而是能够"查证—执行—汇总"，以保证时效性与可控性[3]，[4]。第二类是自动化需求。典型场景包括报表生成、数据清洗、代码调试、工作流编排等。目标是"给任务，不盯过程"，模型需要能够分解任务并驱动具体操作，而非只生成说明性文本[5]。第三类是多步决策需求。复杂任务往往包含探索、规划、试错和修正等环节，如自动化软件开发、科学实验设计、业务流程优化。而单轮问答无法承载"立目标—出方案—执行—根据反馈调整—结束"的闭环过程[2]。

在此背景下，LLM 不再只是一个"函数调用的黑盒模型"，而演化为一个与环境交互、可调用外部能力、具备多步规划与执行逻辑的系统，即 LLM agent。其关注点从"参数和训练数据"扩展到"记忆、工具、环境、调度、监控"等更加宏观且复杂的系统要素。LLM agent 更适合被看作"以 LLM 为核心决策模块的人机—机机协作系统"，而不是简单的"更强的聊天机器人"。

1.2 LLM agent 的基本概念与典型架构

1.2.1 LLM agent 的定义与特征

在本文语境下，LLM agent 相比纯 LLM，更加强调"闭环"、"目标导向"和"可操作"。可以用一个抽象的循环来刻画典型 agent 的工作方式："Perception – Reasoning – Action – Feedback"[3]。

感知阶段从外部环境接收信息，包括用户输入（自然语言指令、表单配置等）、环境状态（网页内容、文件系统、数据库快照、API 返回）以及历史上下文（对话记录、长期记忆、日志）等。推理阶段利用 LLM 对当前状态进行理解和规划，典型工作包括任务理解与重述、任务分解与子目标规划、工具选择与参数构造以及中间思考过程（自反思）等。行动阶段基于决策结果对环境施加操作，如调用工具或 API（搜索、RAG、代码执行、企业内部服务）、操作软件环境（浏览器、IDE、工作流引擎、机器人）或与人类协作（请求确认、抛出选项、发起审批）等。反馈阶段观察行动产生的结果并更新内部状态，包括工具返回、API 响应、运行日志、任务进度（哪些子任务已完成或失败）以及来自人类的纠正、确认或新约束等。

在上述循环基础上，一个满足"agent"基本要求的系统通常具备以下核心特征[6]。首先是目标导向，系统以"达成目标或完成任务"为中心，而非"回答单个问题"。它允许在执行过程中根据反馈动态调整路径，而不是一次性生成全文答案，能根据执行反馈持续修正行为，而不是"生成后就结束"。系统支持多轮环境感知与行动，直到达到终止条件（目标完成、次数或时间上限、失败策略触发等）。其次是多步决策与规划能力。内部显式或隐式存在"先做什么、再做什么"的决策结构，体现为任务分解、迭代试错、回滚与重试、策略切换等行为模式。第三是工具与环境依赖，不将 LLM 视为"万能文本生成器"，而是通过工具调用、外部知识源、执行环境来扩展能力边界。模型行为受环境状态约束，网页结构、API 速率限制、文件系统权限等都会影响决策。

第四是状态与记忆机制。系统拥有跨轮交互的"内部状态"，对已完成工作、失败尝试、用户偏好等进行记录和利用。这可能包含短期会话记忆（上下文窗口内）、外部长期记忆（向量库、数据库）以及结构化任务状态（task graph 或 workflow DAG）。最后是可控与可监控性，行为具备可解释结构（如显式 plan、tool call、log），便于监控与调试，并支持人类在环（可介入审批关键步骤、修正规划、限制权限）。

总结来说，LLM agent = LLM（决策）+ 工具与环境（执行）+ 状态与记忆（上下文）+ 调度与监控（控制），其本质是一个围绕目标、在环境中运行的智能决策系统，而不只是一次性文本映射函数[2]。

1.2.2 典型 Agent 架构模式

围绕上述循环，不同工作在“如何组织感知-推理-行动”上出现了相对主流的架构模式。下面按“从简单到复杂”梳理几类常见范式，并简述其适用场景与局限。

1.2.2.1 ReAct / Tool-calling：推理-行动交替的单体 Agent

这是目前工业界最主流、也是接入门槛最低的 agent 形态，大量产品在“chat + tools”的范式下实现了首批自动化能力[7]。核心思想是将“思考”与“行动”显式交替，利用 LLM 同时完成推理与工具调用决策。典型模式有

- ReAct（Reason + Act）：LLM 先生成中间思考（thought），再决定调用哪个工具、用什么参数（action），获取观察结果（observation），继续新一轮 thought -> action。
- OpenAI-style tool calling / function calling：以结构化 schema 定义可用工具，LLM 在生成时决定是否调用某个函数，并给出参数，系统执行后将结果回填给 LLM，形成多轮对话式交互[8]。

这种架构的架构简单，通常是“单个 LLM + 工具注册表 + 执行器”的单体智能体，并且没有显式全局规划模块，规划隐含在每一步的 Thought 中。因此更适合中小规模任务，如问答增强搜索、简单数据查询与处理、单次或少量工具调用的场景。优点是实现成本低、调试直观、与现有对话接口兼容性好，缺点同样明显：对长程规划、复杂依赖任务支持有限，容易出现“局部贪心”“反复试错但不收敛”的行为。

1.2.2.2 Planner–Executor / Multi-module：显式规划与分工

核心思想是将"规划"（Plan）与"执行"（Execute）解耦，通过多个子模块协同提高复杂任务的可控性和完成率[9]。基本结构包括规划器、执行器和可选的调度器三个核心模块。

规划器负责对全局目标进行任务分解，生成任务树、步骤列表或工作流图[10]。它输出可执行的子任务描述，每个子任务可能绑定不同工具或子 agent[11]。Planner 会分析根任务，查询可用工具清单，确定是直接执行还是分解为复合或原子子任务，并生成层次任务网络（HTN）编码任务层级及其时序和因果依赖关系[12]。执行器负责逐步执行 Planner 产生的子任务，包括调用工具、处理结果、更新状态[11]。每个原子子任务会触发动态实例化，配置专门工具集并在指定系统消息下运行。Executor 必须与 Planner 解耦，确保 Planner 崩溃不影响工具，工具崩溃也不影响 Planner[11]。调度器作为可选组件，决定何时重新规划、如何处理执行失败、如何在多个子任务间调度资源[13]。部分架构中的重规划器会分析执行结果，检测执行循环，必要时生成新计划[14]。Orchestrator 确保任务按正确顺序运行，处理依赖关系并从错误中恢复[11]。

这种架构更适合长流程、多依赖的任务，如端到端应用开发、复杂报表与数据管道构建、多系统集成工作流，或者需要明确"过程可见"、"可插入人工审批"的企业级场景[10]。优点是结构清晰，便于监控和干预，可针对 Planner 与 Executor 分别优化，在整体性能和鲁棒性方面优于现有方法[10]。缺点是规划与执行之间可能出现"计划不落地"或"计划陈旧"问题，需要反复重新规划。系统复杂度上升，对工程与评测提出更高要求，且可能因冗余通信或资源分配不均导致性能下降。模块化架构支持将子计划作为可开发、调试和重用的实体，实现"计划库"和新工作流的敏捷开发[15]。

1.2.2.3 Memory-augmented Agents：显式记忆与长期上下文

显式记忆与长期上下文的核心思想是为 Agent 配备显式的记忆系统，使其能够存储、检索和利用历史信息，突破单次对话上下文窗口的限制，实现跨会话的知识积累和个性化适应。记忆系统通常分为多个层次。

短期记忆对应当前对话或任务的上下文，通常直接保存在 LLM 的 prompt 中，用于维持对话连贯性和任务状态。随着对话进行，短期记忆可能因上下文长度限制需要压缩或转移。长期记忆则是持久化存储的知识库，包括历史对话、执行过的任务、学习到的经验、用户偏好等。实现方式通常采用向量数据库（如 Pinecone、Weaviate）进行语义检索，或使用结构化数据库存储事实性知识。长期记忆支持跨会话的知识复用，使 Agent 能够"记住"用户习惯、避免重复错误。情景记忆记录具体事件和执行轨迹，类似人类的"经历回忆"。Agent 可以通过检索相似历史情景来指导当前决策，例如"上次遇到类似问题时采用了什么方案"。语义记忆则包含抽象的概念和规则知识，如领域知识图谱、API 使用规范、业务逻辑等。这部分记忆往往通过知识蒸馏、总结提取等方式从情景记忆中归纳而来。

记忆的使用通常涉及三个关键操作：存储将新信息编码并写入记忆系统，检索根据当前任务需求查询相关记忆，更新则根据新经验修正或扩展已有记忆。先进的架构还会引入记忆管理机制，如重要性评分、遗忘机制、记忆整合等，防止记忆库无限膨胀或充斥噪声。

这种架构适合需要长期交互、个性化服务的场景，如个人助手、客户服务机器人、持续学习的研发助手等。优点是能够实现真正的"有记忆"智能体，提升用户体验和任务连续性，并支持从历史中学习，减少重复错误。缺点是记忆检索的准确性和效率直接影响性能，不相关记忆可能干扰决策。记忆系统的维护成本高，需要处理隐私、数据一致性等工程问题，而如何有效组织和利用海量记忆仍是开放性研究课题。

1.2.2.4 Multi-agent 系统：协同与分布式智能

协同与分布式智能的主旨是将复杂任务分解给多个专门化的 Agent，通过协作、竞争或协商机制完成单个 Agent 难以胜任的工作。每个 Agent 可以有不同的角色定位、工具配置、知识背景，甚至使用不同的 LLM 后端。Multi-agent 系统的关键在于定义清晰的通信协议和协作范式 [16]。

角色分工型模式为不同 Agent 分配特定职责，如"研究员""编码员""审核员"。各 Agent 按流水线或并行方式协作，每个专注于自己擅长的子任务。例如软件开发场景中，产品经理 Agent 负责需求分析，架构师 Agent 设计系统，开发 Agent 编写代码，测试 Agent 验证质量。

辩论与共识型模式让多个 Agent 对同一问题提出不同方案或观点，通过辩论、投票或评分机制达成共识。这种模式能够利用"思维多样性"减少单一模型的偏见和盲点，提高决策质量。层次协调型模式则设立管理者 Agent 负责任务分配和进度监控，工作者 Agent 执行具体子任务并汇报结果。管理者可以动态调整分工、处理冲突、协调资源，类似企业组织结构。

市场机制型模式中，Agent 之间通过"招标""竞价"等方式争夺任务执行权，系统根据能力、负载、成本等因素选择最合适的 Agent。这种方式适合动态环境下的资源优化配置。Multi-agent 系统的通信方式可以是显式消息传递（如共享消息队列、发布-订阅）、共享工作空间（如白板、共享文档）或通过中心协调器中转[17]。先进系统还会引入元认知层 [18]，让 Agent 能够"理解"其他 Agent 的状态、意图和能力，实现更高效的协同 [19] 。

这种架构适合高度复杂、需要多领域专业知识的任务，如科研辅助、企业级自动化、大规模系统运维、复杂游戏或模拟环境等。优点是可扩展性强，易于并行化和分布式部署，通过专业化分工提升整体能力边界，并支持模块化开发和渐进式迭代。缺点是协调开销显著，通信成本和延迟可能成为瓶颈。Agent 之间可能出现冲突、死锁或低效协作，系统调试和故障定位难度大，而如何设计有效的激励机制和协作协议仍需深入探索。部分研究还关注 Agent 社会动力学，如涌现行为、联盟形成、知识扩散等现象，为构建更智能的群体系统提供理论基础。

1.3 两个核心问题

围绕 LLM agent，当前研究与应用中存在两个突出问题：

1.3.1 如何系统评测大模型智能体?

现有评测多数沿用模型评测思路（例如单轮QA、代码题、静态基准），难以覆盖真实 agent 行为中的任务分解、工具使用、长期记忆、多轮修正等关键环节[20]。即使有专门的 agent benchmark，也往往关注单一维度（如网页浏览、代码执行、游戏环境），评测设置和指标口径差异较大，横向比较困难[21]。缺乏一个从“任务—行为—系统”多层面出发、能够统一描述不同 agent 工作的评测视角。

1.3.2 这些能力究竟由哪些因素共同决定?

实际应用中，agent 的表现不仅取决于底层 LLM 的能力，还受到（a）任务类型与环境复杂度（b）工具选择与接口设计（c）规划 / 反思 / 记忆等决策机制[7]（d）系统工程因素（容错、监控、人类在环策略）等多方面影响。现有很多工作只给出“某配置比某配置更强”的经验结论，缺少系统化的因素分解与结构化描述[22]。

本文的目标，是围绕这两个问题，提供一个简洁但可操作的框架：一方面，从任务、行为、系统三个层面梳理主要评测维度；另一方面，分析影响 agent 能力的关键因素及其相互作用，为后续设计、调优和对比不同 LLM agents 提供参考视角。

2 大模型智能体的评测维度

2.1 核心能力（LLM理论相关）

在当前的Agent研究中，"规划与推理"、"工具使用与执行"以及"记忆管理与上下文一致"这三项能力并非彼此独立，而是相互交织的能力簇 [6]。它们在实际系统中紧密配合，共同决定Agent的表现。本节将在原有框架基础上进一步展开，从抽象能力深入到具体的技术实现路径、可量化的评价指标，以及实践中已知的失效模式。

2.1.1 规划与推理

2.1.1.1 范式与实现机制

基于大语言模型的规划与推理可以分为四类核心范式。每类范式的评测需要采用不同的度量标准，而不能简单地用"正确率"一刀切。

隐式链式推理（implicit Chain-of-Thought）REF 是通过在海量训练语料中暴露大量多步推理样本，让模型将推理结构"内化"到参数空间中。这类推理在生成答案时通常采用端到端的方式，直接输出最终结果，中间的推理轨迹对开发者是不可见的。模型的训练信号主要来自下一个token的似然最大化，或者通过监督微调（SFT）[23] 和人类反馈强化学习（RLHF）[24] 获得。这种方式的问题在于，我们只能从输入和输出之间的映射关系间接推断模型的推理能力，难以将其作为显式的规划组件应用于Agent系统 [25]。如果我们记录任务成功率与推理深度的关系，可以观察到成功率 $\mathrm{Acc}_{\mathrm{implicit}}(d)$ 会随着推理深度 d 的增加而快速下降，而且这个下降趋势很难调控。

外显链式推理（explicit Chain-of-Thought）[26] 则通过精心设计的提示词或系统约束，强制模型输出中间推理步骤。典型方法包括标准CoT、自洽性验证（self-consistency）、逐步推理（step-by-step）以及从简到难（Least-to-Most）等策略。在Agent框架中，这些中间步骤可以近似看作执行计划，为后续的工具调用和决策提供依据。

在轨迹采样策略上，我们需要在单轨迹贪心解码（temperature接近0）和多轨迹采样投票之间做权衡。后者在数学推理等任务中可以将准确率提升约15%到20%，但推理成本和延迟会随着采样数 k 线性增长，即总成本为：

$$\mathrm{Cost} = k \cdot C_{\mathrm{base}} \tag{1}$$

对于生成的推理轨迹，我们可以进行压缩后存入外部记忆库，供后续子任务复用。复用带来的价值可以量化为节省的时间：$\mathrm{ReuseGain} = T_{\mathrm{rerun}} - T_{\mathrm{reuse}}$，其中 T 表示执行时间。

错误定位通常采用反思式提示（reflective prompting）[27] 或自我批评（self-critique）[28] 机制，让模型对推理步骤打分，标记出可疑的环节后进行局部重采样。不过，修正成功的概率会随着错误出现的深度增加而降低，这个关系可以表示为：

$$P_{\mathrm{fix}} \propto e^{-\lambda d} \tag{2}$$

其中 λ 是衰减系数，d 是错误所在的推理深度。

结构化推理与搜索方法 [29] 将大语言模型的输出看作搜索空间中的节点，通过深度优先、广度优先或蒙特卡洛树搜索（MCTS）等算法在"思维树"上寻找最优解。这类方法的核心挑战在于如何给节点打分以及如何分配搜索预算。

节点的评分可以综合多个来源：模型的自我评分、辅助的奖励模型评分，以及任务特定的评分函数，形成一个复合的价值估计：

$$\hat{V}(s) = \alpha V_{\text{LLM}}(s) + \beta V_{\text{RM}}(s) + \gamma R_{\text{task}}(s) \tag{3}$$

其中 α、β、γ 是权重系数。

搜索预算的分配 [30] 需要在token数量和时间约束下自适应地决定搜索深度和分支因子。我们要避免在简单任务上浪费资源，同时也要防止在复杂任务上探索不足。

在Agent场景中，推理树的节点可能对应真实的环境调用（比如API请求或代码执行），这会引入不确定性。当某个调用失败时，需要实施回溯和重新规划。节点失败导致回溯的概率可以建模为：

$$P_{\text{backtrack}} = 1 - \prod_{i=1}^{n}(1 - \epsilon_i) \tag{4}$$

其中 ϵ_i 是第 i 个节点的失败概率。

明确任务规划在Agent框架中指的是将高层任务分解为一系列可执行的子任务。这个过程常常伴随着领域特定语言（DSL）的使用，比如伪代码、工作流DSL或者PDDL规范 [31] 。任务分解可以采用自顶向下的分层策略，也可以采用增量式规划。计划的表示形式会直接影响其可验证性和可执行性。典型的表示方式包括自然语言、JSON结构、Python风格的伪代码或者状态机规范等。评测计划的质量不应该只看任务是否完成，还要考察分解的质量和执行的效率。例如在复杂的网页操作任务中，Agent在相同工具集下的表现可以通过工具调用次数 N_{tool} 和总token消耗 C_{total} 来量化。更优的策略应该能够减少工具调用次数（$\Delta N_{\text{tool}} < 0$）同时降低token消耗（$\Delta C_{\text{total}} < 0$）。

2.1.1.2 评测指标与失效模式

除了任务级别的成功率，过程性指标更能揭示规划能力的本质 [32]。

计划冗余度 [33] 衡量的是计划中冗余步骤的占比：

$$\rho_{\text{redun}} = \frac{N_{\text{redundant}}}{N_{\text{total}}} \tag{5}$$

冗余度过高通常表明模型存在"过度解释"的倾向，生成了许多不必要的步骤。

计划可执行性由语法错误率和参数错误率共同决定：

$$\text{ExecRate} = 1 - \frac{E_{\text{syntax}} + E_{\text{param}}}{N_{\text{plan}}} \tag{6}$$

这个指标直接暴露了大语言模型在schema对齐上的局限性。

轨迹一致性 [34] 度量的是多次采样下关键决策点的重合程度：

$$\text{Consist} = \frac{|D_{\text{intersection}}|}{|D_{\text{union}}|} \tag{7}$$

如果一致性过低，说明模型的随机性太大，这会损害系统的可预测性和工程可维护性。

错误恢复能力通过失败后成功调整的比例来衡量：

$$\text{Recovery} = \frac{N_{\text{success-after-fail}}}{N_{\text{fail}}} \tag{8}$$

在实践中，规划失效主要表现为以下四类模式：

- 伪规划 [35] 指的是模型生成了看似合理但实际上缺乏因果关联的多步计划，或者计划与真实工具的能力不匹配。例如，模型可能规划了需要调用某个API，但该API实际上并不存在或无法完成预期功能。
- 过度分解与欠分解是两个极端。过度分解是指简单任务被拆解成许多无意义的子步骤，增加了不必要的复杂度；欠分解则是复杂任务只得到粗略的框架，导致执行时频繁需要返工和重新规划。
- 目标漂移 [36] 在长对话和开放任务中尤为显著。在执行过程中，如果出现新的推理分支，模型可能会逐步偏离原始目标。目标漂移的程度可以通过目标嵌入向量的余弦相似度来量化：

  $$\text{Drift} = 1 - \cos(\mathbf{v}_{\text{original}}, \mathbf{v}_{\text{current}}) \tag{9}$$

  其中 $\mathbf{v}_{\text{original}}$ 是原始目标的向量表示，$\mathbf{v}_{\text{current}}$ 是当前目标的向量表示。

2.1.2 工具使用与执行

2.1.2.1 接口与协议设计

大语言模型与工具的交互已经逐渐收敛到function calling模式。这种模式的设计要素包括schema设计、编码解析和错误处理[37]。

在schema设计上，需要平衡自然语言描述的易理解性和形式化类型约束的严格性。一个好的函数签名应该包含函数名称、功能描述、参数列表以及返回值规范。参数的约束可以通过JSON schema或正则表达式来定义取值空间，这样可以支持静态验证和自动修正。实验表明，描述的质量对调用成功率有显著影响。在函数描述中添加负面示例（即什么情况下不应该使用这个函数）可以使调用准确率 Acc_{call} 提升约8到12个百分点。

在编码与解析方面，需要在纯JSON API和对大语言模型友好的文本格式之间做权衡。纯JSON输出有利于结构化解析，但要求模型在训练或监督微调阶段强化"严格JSON"的生成分布，确保即使在高温度采样下也能保持语法的合法性。

流式模式（Streaming mode）支持在单次回答中进行多轮工具调用 [38]，形成"工具-模型-工具"的循环。这种模式的吞吐量可以量化为：

$$\text{Throughput} = \frac{1}{L_{\text{LLM}} + L_{\text{tool}}} \tag{10}$$

其中 L_{LLM} 是模型生成的延迟，L_{tool} 是工具执行的延迟。

错误处理策略需要区分不同类型的错误：HTTP错误、schema不匹配以及运行时异常。可以设计一个wrapper层来实施自动参数修正，或者将错误信息返回给模型触发反思和重试。后者的成功率通常在60%到70%之间，即 $P_{\text{retry}} \approx 0.6 - 0.7$。

2.1.2.2 选择与调用策略

工具选择面临着规模扩展的挑战。当工具集的大小 $|\mathcal{T}|$ 超过50个时，将所有工具的描述都注入到上下文中会超出窗口限制，因此需要采用两阶段的检索式选择策略 [39]。

向量检索将工具描述和示例调用映射到embedding空间，基于查询的相似度筛选出候选集 $\mathcal{T}_{\text{cand}} \subset \mathcal{T}$ 。策略模型（policy model）可以作为轻量级的排序器，根据上下文预测各个工具的使用概率 $P(t|\mathbf{x})$ 。

大语言模型自路由（LLM self-routing）通过专门的router prompt来决定是直接回答用户问题，还是调用某一类工具 [40]。路由的准确率需要满足 $\text{Acc}_{\text{route}} > \tau$，通常阈值 τ 需要在0.85以上才能保证系统的可靠性 [41]。而在调用策略上，需要平衡探索与效率。单工具调用与多工具pipeline的差异体现在执行图的复杂度上。复杂任务往往需要生成显式的有向无环图（DAG）结构，由执行器（executor）来编排执行。探索式调用先通过探测型调用获取环境信息，再规划核心步骤。这种方式的鲁棒性优于目标导向式，但需要确保信息增益与成本的比值满足一定条件：

$$\frac{\text{Gain}_{\text{info}}}{C_{\text{probe}}} > \theta \tag{11}$$

其中 θ 是预设的阈值。预算感知调用要求Agent估计每次工具调用的价值 V_{call}，避免对简单查询触发昂贵的API消耗。价值模型可以训练为：

$$V_{\text{call}} = f(\mathbf{x}, \text{complexity}(\mathcal{T})) \tag{12}$$

其中 $\mathbf{x}$ 是输入，$\text{complexity}(\mathcal{T})$ 是工具的复杂度。

2.1.2.3 执行细节与评估

参数填充与schema对齐是执行层面的关键挑战 [42]。槽位填充（slot filling）需要从自然语言中抽取结构化参数，在多语言环境下准确率会下降约15%到20%。schema演化要求模型在不重新微调的情况下适应接口的变更，其适应速度可以用准确率随时间的变化率 $\frac{\Delta \text{Acc}}{\Delta t}$ 来衡量。

调用结果整合需要处理引述式解释和归纳式总结之间的权衡。当工具返回的结果包含噪声或存在矛盾时，模型应该表达出不确定性。这种不确定性的估计可以与结果的熵相关联：

$$H = -\sum_{i} p_i \log p_i \tag{13}$$

其中 p_i 是各个可能结果的概率。

在评测指标上，应该区分不同的失败来源。工具调用成功率可以分解为三个部分：语法正确率 $\text{Acc}_{\text{syntax}}$、参数合法性 $\text{Acc}_{\text{param}}$ 和无异常率 $\text{Acc}_{\text{except}}$。端到端的成功率是这三者的乘积：

$$\text{Succ}_{\text{e2e}} = \prod_{i} \text{Acc}_i \tag{14}$$

工具必要性比率衡量的是在最优策略下的调用效率：

$$\eta = \frac{N_{\text{needed}}}{N_{\text{actual}}} \tag{15}$$

理想情况下，η 应该接近1，表示没有不必要的工具调用。此外，还需要分别统计子任务成功率和语言生成错误率，以便定位系统的瓶颈所在。

常见的失效模式包括三类 [43]：

- 工具幻觉：模型捏造了不存在的工具或参数。在开放域任务中，这种幻觉的发生概率可以达到5%到10%，即 $P_{\text{hallu}} \approx 0.05 - 0.1$。
- 工具箱误用：选择了错误的工具（例如用翻译API去处理数学问题），或者忽略了高效的专用工具而直接用语言推理来解决问题。

- 无边界重试：发生错误后，模型不修改参数而重复调用同一个工具，陷入循环。这种循环的长度分布服从重尾分布 $P(L > l) \sim l^{-\alpha}$，即存在极端长的循环。

2.1.3 记忆管理与上下文一致

2.1.3.1 记忆类型与实现架构

在大语言模型Agent中，记忆通常被显式划分为不同的存储模块，每个模块对应不同的数据结构和检索策略 [44]。

短期工作记忆对应当前对话的上下文，由框架负责剪裁和组织 [45] 。常见的做法是保留最近 K 轮对话，或者基于显著性评分 $s(\mathbf{x})$ 动态选择。显著性函数可以设计为：

$$s(\mathbf{x}) = \lambda_1 \text{len}(\mathbf{x}) + \lambda_2 \text{ent}(\mathbf{x}) + \lambda_3 \text{imp}(\mathbf{x}) \tag{16}$$

综合考虑长度、信息熵和重要性信号，其中 λ_1、λ_2、λ_3 是权重系数。

长期语义记忆存储用户画像、偏好摘要和历史经验。技术上通常采用向量数据库（如FAISS、HNSW）配合embedding模型实现相似性检索 [46]。检索的质量依赖于embedding空间的质量，可以通过记忆召回率来度量：

$$\text{Recall}_{\text{mem}} = \frac{|\mathcal{R} \cap \mathcal{R}^*|}{|\mathcal{R}^*|} \tag{17}$$

其中 $\mathcal{R}$ 是检索到的记忆集合，$\mathcal{R}^*$ 是所有相关记忆的全集。更高阶的方案采用层次化的记忆结构，先检索粗粒度的摘要，再逐级深入到细节（drill-down）。这种方式的检索延迟可以表示为：

$$L_{\text{hier}} = L_{\text{coarse}} + p_{\text{hit}} \cdot L_{\text{fine}} \tag{18}$$

其中 L_{coarse} 是粗检索的延迟，p_{hit} 是粗检索的命中率，L_{fine} 是细检索的延迟。

程序性记忆记录任务执行的日志和验证方案，采用结构化日志加事件溯源（event sourcing）的方式存储，并按任务类型、工具签名和错误码建立索引。查询效率对任务复用至关重要，索引的命中率应该满足 $\text{HitRate} > 0.9$ 才能保证实用价值。

2.1.3.2 写入、压缩与更新机制

记忆写入策略需要在即时性和系统负载之间做权衡 [45] 。写入可以基于显著性阈值 τ_{salience} 触发，即只有当 $s(\mathbf{x}) > \tau$ 时才激活写入操作。写入形式的选择直接影响存储效率。原文存储的压缩比为1，而摘要存储可以达到：

$$\rho_{\text{comp}} = \frac{L_{\text{abs}}}{L_{\text{orig}}} \approx 0.1 - 0.2 \tag{19}$$

其中 L_{abs} 是摘要长度，L_{orig} 是原文长度。

采用异步后台处理可以降低交互延迟约30%（$\Delta L_{\text{interact}} \approx 30\%$），但会引入一致性问题，需要通过版本向量 $\mathbf{v}_{\text{version}}$ 来解决。

记忆压缩通过静态聚类和动态摘要实现。定期聚类将历史事件压缩为高层表示，压缩误差可以量化为：

$$\epsilon = ||\mathbf{e}_{\text{cluster}} - \mathbf{e}_{\text{centroid}}|| \tag{20}$$

其中 $\mathbf{e}_{\text{cluster}}$ 是簇内事件的向量表示，$\mathbf{e}_{\text{centroid}}$ 是簇中心。

动态摘要在检索后实施，避免上下文过载。可以采用层次结构组织，分为session级、topic级和global级。检索时根据查询的性质 q 确定检索的层级：

$$l^* = \arg\max_l P(\text{relevant}|q, l) \tag{21}$$

记忆更新与遗忘机制需要处理过时信息和冲突。时间衰减模型可以降低旧信息的权重：

$$w(t) = w_0 \cdot e^{-\lambda(t-t_0)} \tag{22}$$

其中 w_0 是初始权重，λ 是衰减系数，t 是当前时间，t_0 是记忆产生的时间。

冲突解决可以基于时间戳、来源可信度 c_{source} 和用户的显式确认。隐私合规要求支持按用户清除（per-user清除）和访问控制，存储架构需要满足数据分域约束，即不同用户的数据不重叠：$\mathcal{D}_i \cap \mathcal{D}_j = \emptyset$。

2.1.3.3 检索、RAG、构建与一致性保障

检索策略通常采用多通道融合来提升覆盖率[47]。在向量检索之后，可以实施重排序（reranking），相关度的重排函数可以训练为：

$$\text{score}(\mathbf{x}, \mathbf{m}) = \mathbf{w}^T \phi(\mathbf{x}, \mathbf{m}) \tag{23}$$

其中 $\mathbf{w}$ 是权重向量，$\phi(\mathbf{x}, \mathbf{m})$ 是特征函数。

多通道检索对用户画像、任务日志和文档采用不同的检索器，最终融合结果为：

$$\mathcal{R}_{\text{final}} = \bigcup_{c \in \mathcal{C}} \mathcal{R}_c \tag{24}$$

其中 $\mathcal{C}$ 是所有检索通道的集合。

查询重写（Query重写）通过大语言模型生成检索友好的查询变体，可以提升召回率约10%到15%（$\Delta\text{Recall} \approx 10 - 15\%$）。

上下文构建需要在token预算 B 的约束下优化信息密度。选择策略可以建模为一个背包问题：

$$\max_{\mathcal{M} \subset \mathcal{R}} \sum_{m \in \mathcal{M}} \text{info}(m) \quad \text{s.t.} \quad \sum_{m \in \mathcal{M}} L(m) \leq B \tag{25}$$

即在token总量不超过 B 的情况下，最大化信息量。

一致性约束可以通过显式标注"最新信息"和"历史记录"来辅助模型判断，标注的准确率会影响上下文利用率 η_{context}。链式记忆引用采用ID或链接的形式，便于执行层进行溯源验证。

在评测指标上，我们关注记忆的有效性。记忆召回准确率可以通过合成任务来测试，例如提问"上周预订的餐馆名称"，期望答案与记忆库中事实的精确匹配率应该达到95%以上（$\text{Acc}_{\text{recall}} > 0.95$）。上下文一致性通过长对话中角色设定和任务状态的保持率来衡量。自动化评测可以采用规则匹配或embedding相似度 $\cos(\mathbf{v}_{\text{gt}}, \mathbf{v}_{\text{gen}})$，其中 $\mathbf{v}_{\text{gt}}$ 是真实值，$\mathbf{v}_{\text{gen}}$ 是生成值。记忆污染鲁棒性测试通过注入对抗样本来评估，理想的系统应该将错误记忆持久化的概率控制在5%以下（$P_{\text{persist}} < 0.05$）。

记忆失效主要表现为以下三类模式 [48]：

- 记忆灾难性干扰指的是无关的记忆片段主导了决策，导致性能显著下降（$\Delta\text{Acc} > 20\%$）。
- 伪记忆（Confabulation）是模型基于语言先验虚构的交互历史，并不是真实发生过的。在长对话中，虚构的概率可以达到10%到15%（$P_{\text{confab}} \approx 0.1 - 0.15$）。
- 时间错置表现为混淆历史状态和当前状态。例如，使用过时偏好的频率可以量化为：

$$\text{Rate}_{\text{stale}} = \frac{N_{\text{stale}}}{N_{\text{total}}} \tag{26}$$

理想情况下应该低于0.02，即2%以下。

2.1.4 跨维度综合考量

在实际的Agent系统中，规划、工具使用和记忆管理是强耦合的 [49]。大多数失效情况源于系统协同的缺陷，而不是单一模块的问题。因此，贴近实际的评测应该采用交互式环境，结合过程日志分析。

例如，在网页代理任务中，除了任务成功率，还应该考察决策链条的成本：

$$\text{Cost}_{\text{chain}} = C_{\text{token}} + \mu \cdot N_{\text{tool}} + \nu \cdot T_{\text{latency}} \tag{27}$$

其中 C_{token} 是token消耗，μ 是工具调用的单位成本，N_{tool} 是工具调用次数，ν 是延迟的单位成本，T_{latency} 是总延迟。在相同成功率下，成本更低的系统具有更高的实用价值。

可解释性与可重放性要求日志能够支持完整的决策过程复现 [50]。对抗鲁棒性测试会故意构造误导信息、有噪声的工具返回值或环境变更（例如API schema从版本 v_i 升级到 v_{i+1}）。理想的Agent应该能保证有界的性能退化 [51]：

$$|\text{Acc}_{\text{new}} - \text{Acc}_{\text{old}}| < \delta \tag{28}$$

通常 δ 取0.1，即性能下降不超过10%。

未来的研究方向可以包括：

- 显式引入world model来提升长期规划的稳定性
- 通过离线强化学习在真实交互日志上优化工具策略
- 在记忆系统中引入知识图谱以增强冲突检测能力

这些方向要求评测基准从静态的问题集转向动态的环境评估，最终形成"交互式环境 + 过程诊断 + 成本建模"的三维评测体系。

2.2 系统与工程（并非不核心）

在核心能力维度之外，大模型智能体的"系统与工程"侧能力直接决定其在真实场景中的可用性与可维护性 [52]。这一部分在一些研究中被低估为"非核心"，但从系统视角看，它们与模型内在能力同等关键，且可被相对精确定义与度量 [53]。

2.2.1 系统性能与效率

2.2.1.1 延迟性能指标

端到端响应时间（End-to-End Response Time）是衡量Agent系统最直观的性能指标 [54]，定义为从用户输入查询到系统返回完整响应的总时间。在实际部署中，我们通常采用百分位数指标来刻画系统延迟分布。对于包含N次请求的样本集合，第p百分位延迟L_p定义为：

$$L_p = \inf\{t \mid P(T \leq t) \geq \frac{p}{100}\} \tag{29}$$

其中T表示随机变量响应时间。

P99延迟 [55] 具有特殊意义，它表示99%的用户请求能够在该时间阈值内得到响应 [56]。相比算术平均值，P99能够有效捕捉长尾延迟问题，这对于用户体验至关重要。在高并发场景下，即使只有1%的慢请求，在每秒千次请求的系统中也意味着每秒有10个用户遭遇糟糕体验。

首token时间（Time To First Token, TTFT）[57] 是流式输出场景中的关键指标，定义为从请求发起到接收到第一个生成token的时间间隔 [56]。该指标直接影响用户对系统响应性的感知。对于采用自回归解码的LLM Agent，TTFT主要由以下几个部分组成：

$$\text{TTFT} = T_{queue} + T_{prefill} + T_{network} \tag{30}$$

其中T_{queue}为请求排队等待时间，$T_{prefill}$为prompt预填充计算时间，$T_{network}$为网络传输延迟。在多数情况下，预填充时间占主导地位，其复杂度与输入序列长度成正比。优化TTFT的P99指标对于提升交互体验具有重要意义，特别是在需要快速反馈的对话场景中。

2.2.1.2 并行化效率

现代LLM Agent系统常支持并行执行多个独立工具调用 [58]。并行化效率η_{parallel}可以定义为：

$$\eta_{\text{parallel}} = \frac{T_{\text{sequential}}}{T_{\text{parallel}}} \tag{31}$$

其中$T_{sequential}$为串行执行总时间，$T_{parallel}$为并行执行总时间。理想情况下，n个完全独立的操作并行化效率应接近n。然而实际系统受限于资源竞争、同步开销等因素，P99并行化效率往往显著低于理论值，这要求系统设计时需要仔细平衡并行度与资源利用率。

2.2.1.3 可扩展性与稳定性

系统吞吐量（每秒处理请求数）与延迟存在固有权衡关系。根据Little's Law [59]，系统中的平均请求数L、到达率λ和平均响应时间W满足：

$$L = \lambda W \tag{32}$$

在负载增加时，P99延迟通常呈非线性增长。我们定义临界负载$\lambda_{critical}$为使P99延迟超过服务质量目标的最小到达率。监控不同负载下的P99延迟曲线，有助于确定系统的容量规划参数和自动扩缩容策略。

P99延迟的优化往往比优化平均延迟更具挑战性。常见的长尾延迟来源包括：垃圾回收停顿、资源竞争、网络抖动和模型推理的输入长度敏感性 [60]。针对LLM Agent系统，可以考虑的优化方向有：采用流式生成减少首字延迟、实施智能重试机制处理超时请求、使用推测执行预测性地准备资源，以及通过对冲请求（hedged requests）降低尾部延迟的影响。

2.2.2 任务完成度与输出质量

在实际系统中，"任务是否完成"通常不等价于"模型是否给出一个看似合理的答案"。工程上更关心可操作的成功定义、可比较的指标以及与成本的关系 [61]。

2.2.2.1 任务完成度定义与指标

任务完成度需要从环境视角进行客观评估。例如，当智能体被要求完成在线预订酒店的任务时，成功与否不应只看它是否生成了一段看似合理的回复，而应检查预订是否真正完成——是否收到了确认邮件，订单是否出现在系统中等。

任务成功率可以定义为智能体在测试任务集合中实际完成任务的比率 [61]。对于包含多个步骤的复杂任务，还需要考虑阶段成功率和路径效率。例如，在数据分析和可视化任务中，智能体可能成功完成了数据清洗阶段，但在图表生成阶段失败。

路径效率 [61] 衡量智能体完成任务所需的步骤与理论最优步骤的接近程度，这直接关系到执行成本和用户体验。比如，一个高效的智能体应该能用最少的点击和输入完成网页表单填写。

$$\text{PathEfficiency} = \frac{1}{K}\sum_{k=1}^{K}\frac{\text{OptimalSteps}_k}{\text{ActualSteps}_k} \tag{33}$$

其中 K 是任务数量，OptimalSteps_k 是任务 k 的理论最优步骤数，ActualSteps_k 是智能体实际步骤数。

2.2.2.2 输出质量评估

输出质量可以从多个维度进行量化评估：

可执行性关注输出是否能在目标环境中正常运行。例如，在代码生成任务中，这包括语法正确率和测试通过率：

$$\text{TestPassRate} = \frac{1}{N}\sum_{j=1}^{N}\mathbb{1}[\text{tests}(output_j)\ \text{全部通过}] \tag{34}$$

正确性衡量输出内容与预期结果的匹配程度。在表格数据处理任务中，这可能意味着生成的SQL查询是否返回了正确的结果集。一致性评估智能体在相同或相似输入下产生稳定输出的能力。例如，当多次询问同一事实性问题时，智能体应该给出相同的答案。可用性则从最终用户的角度评估输出质量。在生成长篇报告的场景中，可用性可能包括结构清晰度、语言流畅度和信息完整性等维度。

工程实践中，这些质量指标常与资源消耗结合考虑，形成"单位成本质量"指标，如每千token的输出质量或每秒处理能力的质量产出。

2.2.3 可靠性、鲁棒性

2.2.3.1 不确定性估计与置信控制

智能体应具备对自身能力边界的认知。置信度机制让智能体能够识别不确定的情况并采取适当行动 [62]，如寻求澄清或拒绝回答。例如，当用户询问高度专业化的医疗建议时，负责任的智能体应该明确表示自己能力的局限性，而不是提供可能有害的猜测。

低置信率指标可以量化智能体在多少情况下识别到了自身的不确定性 [63]：

$$\text{LowConfidenceRate} = \frac{1}{N}\sum_{i=1}^{N}\mathbb{1}[\text{Conf}(output_i) < \tau] \tag{35}$$

结合错误预测器，系统可以设置拒绝机制，在检测到高风险时主动拒绝执行某些操作，比如当检测到可能违反安全策略的工具调用请求时。

2.2.3.2 对扰动与分布偏移的鲁棒性

在实际部署中，智能体必须能够处理各种非理想情况。鲁棒性体现在面对输入扰动时的稳定性表现。例如，当用户输入包含拼写错误或表达不完整时，鲁棒的系统仍能理解核心意图并给出恰当响应。我们可以区分两种鲁棒性 [64]：对非语义扰动的稳定性（如同义词替换不应改变回答实质）和对关键语义变化的敏感性（如问题核心要素的改变应该导致不同的回答）。

环境噪声下的性能保持能力也是重要考量 [65]。例如，在嘈杂的语音识别环境中，智能体应能维持合理的任务完成率。

$$\text{Robustness} = \frac{1}{L}\sum_{l=1}^{L}\mathbb{1}[\text{Success}(\text{input}_l) = \text{Success}(\text{perturbed\_input}_l)] \tag{36}$$

其中 L 是测试用例数量，$\text{Success}(\text{input}_l)$ 表示原始输入的任务成功情况，$\text{Success}(\text{perturbed\_input}_l)$ 表示扰动输入下的任务成功情况。

2.2.3.3 故障模式与恢复能力

复杂的智能体系统应具备优雅降级和自动恢复的能力 [66] 。重试有界性确保系统不会陷入无限循环；超时有界性防止任务永久挂起；调用有界性控制资源消耗。故障恢复能力可以通过故意注入错误来测试。例如，模拟API服务暂时不可用的情况，观察智能体是否能够通过备用策略或适当的错误处理继续完成任务。

2.2.4 安全性、AI伦理与对齐

2.2.4.1 有害行为与不当内容的度量

安全性评估需要系统化的风险分类和检测机制。不安全输出率量化智能体产生有害内容的频率，如生成歧视性言论或提供危险建议。对于明确的有害请求，拦截率衡量系统成功拒绝提供不当协助的比例 [67]。例如，当用户请求生成虚假信息或恶意代码时，系统应该坚决拒绝。

2.2.4.2 与人类偏好和规范的一致性

智能体的行为应符合人类价值观和社会规范。偏好准确率评估系统输出与人类评判标准的一致性程度 [68]：

$$\text{PrefAcc} = \frac{1}{M}\sum_{k=1}^{M}\mathbb{1}\left[\text{sign}\left(R(x_k, y_k^A) - R(x_k, y_k^B)\right) = \text{human\_pref}_k\right] \tag{37}$$

约束满足率确保智能体遵守明确的指令限制。例如，当要求以特定格式或字数限制生成内容时，系统应该严格遵循这些约束。

2.2.4.3 工程级安全防护

在实际工程中，多层防护机制可以构成纵深防御体系。输入拦截在请求进入处理流程前过滤明显有害的内容；输出筛选对生成内容进行最终检查，捕捉可能漏过的风险；工具使用监控防止对外部资源的滥用 [69]。例如，在金融应用场景中，系统应该拦截未经授权的交易操作请求，即使这些请求以自然语言形式表达。

2.2.4.4 长期互动与记忆相关的伦理风险

随着智能体具备记忆和个性化能力，新的伦理问题随之产生，需要隐私泄露率评估系统在长期交互中保护用户敏感信息的能力。例如，智能体不应在后续对话中无意间泄露前序会话中的个人信息。记忆合规性确保系统能够按要求"遗忘"特定信息，这在数据保护法规如GDPR [70] 下尤为重要。总体而言，"系统与工程"维度的评测需要围绕三类问题给出定量答案：

- 能否稳定完成真实任务；
- 在错误和扰动下是否有界退化并具备恢复能力；
- 在安全与伦理约束下是否仍能保持有用且合规的行为。

这些问题都可以通过环境可验证的信号、任务级日志和专门的安全评估器转化为具体的指标和基准，为后续的模型改进与系统优化提供明确方向。

3 影响大模型智能体能力的关键因素

3.1 大模型内在因素

3.1.1 模型底层架构与参数量

大模型智能体的核心能力在很大程度上受到其底层语言模型架构的影响。不同的架构设计对应着不同的计算效率、表达能力与可扩展性权衡。

传统的Transformer架构通过自注意力机制实现全局信息交互，其计算复杂度为$O(n^2)$，其中n为序列长度 [71]。这一特性使得标准Transformer在处理长上下文时面临显著的计算瓶颈。为突破这一限制，研究者提出了多种架构变体，包括稀疏注意力机制、线性注意力以及Mixture of Experts [72]等。例如，稀疏Transformer通过限制注意力范围将复杂度降至$O(n\sqrt{n})$或$O(n \log n)$，在长文档理解与多轮对话等任务中展现出明显优势。Mixture of Expert架构则通过条件激活部分专家网络，在保持参数总量的同时降低单次推理的计算开销，使得超大规模模型的部署成为可能。

参数量是另一个影响模型能力的关键因素。经验规律表明，模型性能与参数规模、训练数据量之间存在幂律关系，可近似表示为 [73]：

$$L(N) \propto N^{-\alpha} \tag{38}$$

其中L为损失函数，N为参数量，α为幂律指数。这意味着参数量的提升能够带来模型能力的系统性增强，包括更强的语义理解、推理能力以及少样本学习能力。然而，参数规模的增长并非线性转化为性能提升。当参数量超过某一阈值后，边际收益显著递减，同时训练成本与推理延迟呈指数级增长。此外，参数量需与训练语料规模相匹配，否则容易导致过拟合或欠拟合。根据Chinchilla scaling laws [74]，最优的模型训练应当满足参数量与训练token数的特定比例关系，通常建议训练token数约为参数量的20倍 [75]。在实际应用中，7B至70B参数的模型已能够支持多数复杂智能体任务，而更大规模的模型则主要用于需要极强泛化能力或专业知识的场景。

3.1.2 训练数据，训练方法与知识边界

训练数据的质量、覆盖范围与组成结构直接决定了大模型智能体的知识基础与任务适应能力 [76]。高质量的训练语料应当具备广泛的领域覆盖、多样的文本类型以及较低的噪声比例。现代大模型通常采用包含网页文本、书籍、学术论文、代码仓库等多源异构数据的混合语料，其总规模可达数万亿token。数据配比策略对模型性能有显著影响 [77]，例如在面向代码生成任务的模型中，代码数据的占比通常需达到10%至30%以确保充分的编程能力。数据清洗与去重同样关键，研究表明去除重复内容可显著提升模型的泛化性能，避免模型过度记忆特定文本模式 [78]。

训练方法的演进深刻改变了大模型智能体的行为特征。传统的预训练阶段采用自监督学习（SFT）目标，通过最大化似然函数 [79]：

$$\max_{\theta} \sum_{i=1}^{T} \log P(x_i | x_{<i}; \theta) \tag{39}$$

学习语言的统计规律。然而，这种训练方式产生的模型往往在指令遵循、对话交互等方面表现不佳。指令微调通过在精心标注的指令-响应对上继续训练，使模型学会理解并执行用户意图。人类反馈强化学习（RLHF）则进一步引入奖励模型$r_\phi(x, y)$与策略优化目标 [80]：

$$\max_{\theta} \mathbb{E}_{x \sim \mathcal{D}, y \sim \pi_\theta}[r_\phi(x, y)] - \beta \mathbb{D}_{KL}[\pi_\theta || \pi_{ref}] \tag{40}$$

通过人类偏好数据对齐模型输出与人类价值观。例如，GPT-4与Claude等模型均大量使用RLHF技术，使其在拒绝有害请求、保持对话连贯性等方面显著优于仅经过监督微调的模型。

知识边界是大模型智能体面临的固有限制。由于训练数据具有时间截止点，模型无法获知截止日期之后的事实性信息。更严重的是，模型倾向于在知识不足时生成看似合理但实际错误的内容，即"幻觉"现象 [81]。这一问题在长尾知识、专业领域以及需要实时信息的任务中尤为突出。例如，当询问最新的科研进展或特定企业的财务数据时，模型可能编造不存在的引用或数据。

为缓解知识边界限制，检索增强生成（RAG）成为主流解决方案。该方法在生成前先从外部知识库检索相关文档，形式化为 [82]：

$$P(y|x) = \sum_{d \in \text{retrieve}(x)} P(y|x,d)P(d|x) \tag{41}$$

其中d为检索到的文档。

具体实现中，可使用向量数据库存储文档嵌入，通过近似最近邻搜索快速定位相关内容。此外，工具调用机制允许模型主动访问搜索引擎、数据库、计算器等外部资源，在运行时动态获取所需信息。例如，在代码生成任务中，模型可调用代码解释器验证生成代码的正确性，在数学推理中可调用符号计算引擎确保计算精度。这些机制的引入使得智能体能够突破静态知识限制，处理开放域、动态变化的复杂任务。

3.2 系统设计与工程因素

3.2.1 提示词工程与人机交互

提示词工程作为连接用户意图与模型执行的关键桥梁，对智能体的任务完成质量有着决定性影响 [83]。有效的提示词设计需要在明确性、完整性与灵活性之间取得平衡。研究表明，结构化的提示模板能够显著提升模型在复杂任务中的表现。例如，Chain-of-Thought提示通过引导模型逐步展开推理过程，将复杂问题分解为多个中间步骤，形式化为 [84]：

$$y = f(x, \text{CoT}(x)) \tag{42}$$

其中$\text{CoT}(x)$表示推理链提示。这种方法在数学推理、逻辑推理等任务中将准确率提升了20%至50% [85]。Few-shot prompting [86] 则通过在提示中嵌入少量示例，利用上下文学习能力引导模型理解任务格式，其效果可近似为：

$$P(y|x, \{(x_i, y_i)\}_{i=1}^{k}) \tag{43}$$

其中k为示例数量。

角色定义与约束条件的明确表述同样关键 [87]。通过在系统提示中明确智能体的角色、能力边界以及行为规范，可以有效减少不相关输出与潜在风险。比如在客服场景中，提示词可限定"你是一个专业的技术支持专员，只回答与产品相关的问题，对于超出范围的请求应礼貌拒绝并建议转接人工服务"。动态提示策略进一步增强了交互的适应性，根据对话历史与任务进展动态调整提示内容，实现更精准的意图理解与任务规划。

人机交互模式的设计直接影响智能体的可用性与用户体验 [88]。其中多轮对话管理需要智能体维护对话状态、追踪未完成目标以及处理话题转换。对话状态可建模为

$$s_t = g(s_{t-1}, u_t, a_{t-1}) \tag{44}$$

其中s_t为当前状态，u_t为用户输入，a_{t-1}为上一轮智能体动作。

此外澄清机制 [89] 可以使智能体能够在信息不足时主动询问，避免基于错误假设执行任务。例如，当用户说"帮我订票"时，智能体应询问出发地、目的地、时间等必要信息，而非随意测。

3.2.2 工具链设计与集成

工具链的丰富程度与集成质量直接决定了智能体的行动空间与任务边界。现代智能体系统通常配备多种类型的工具，包括信息检索工具、计算工具、代码执行环境、API接口以及数据库访问等[90]。工具调用过程可抽象为感知-规划-执行的循环，形式化为

$$a_t = \pi(s_t), o_t = \text{execute}(a_t), s_{t+1} = \text{update}(s_t, o_t) \tag{45}$$

其中a_t为选择的工具及参数，o_t为工具返回的观测结果。

工具描述的规范化对模型准确调用至关重要。每个工具应包含明确的功能说明、参数定义、返回格式以及使用约束。例如，搜索工具的描述可为"search(query: str, max_results: int) -> List[Document]: 根据查询词检索相关文档，返回至多max_results个结果，每个结果包含标题、摘要与URL" [91]。OpenAPI规范、函数签名等标准化格式能够帮助模型理解工具语义，减少调用错误。工具选择策略则需要模型根据任务需求与工具特性进行推理 [92]。基于ReAct框架 [93] 的方法将推理与行动交替进行，模型首先思考当前需要什么信息或操作，然后选择合适的工具执行，最后根据结果调整后续计划。

工具执行的可靠性与安全性是工程实践中的核心挑战。沙箱隔离机制确保代码执行、系统命令等高风险操作不会影响宿主环境，通常通过容器化技术或虚拟机实现 [94]。参数验证与输入清洗防止注入攻击与非法操作，例如在数据库查询前对SQL语句进行参数化处理。超时控制与资源限制避免工具调用导致系统挂起或资源耗尽。错误处理机制使智能体能够优雅地处理工具失败，例如当API请求超时时自动重试或切换备用方案。工具编排策略允许复杂任务中的多工具协同，比如先用搜索工具获取原始数据，再用代码执行环境进行数据处理与分析，最后用可视化工具生成图表。

3.2.3 多智能体架构与协作机制

多智能体系统通过角色分工与协作突破单一智能体的能力限制，在复杂任务中展现出显著优势。角色专门化策略将任务分解为多个子任务，每个智能体专注于特定领域或功能 [95]。例如，在软件开发场景中可设计产品经理智能体负责需求分析、架构师智能体负责系统设计、开发者智能体负责代码实现、测试工程师智能体负责质量保证 [96]。这种分工使得每个智能体可以使用专门的提示词、工具链甚至底层模型，提升各环节的专业性。

协作模式的设计决定了智能体间的交互效率 [97]。顺序协作模式中，智能体按固定顺序传递任务，适用于流程明确的场景，可建模为：

$$y = f_n(f_{n-1}(\dots f_1(x))) \tag{46}$$

并行协作模式允许多个智能体同时处理不同子任务，最后汇总结果，适用于可分解的独立任务。层次化协作引入管理者智能体负责任务分配与结果整合，工作者智能体执行具体任务，形成类似组织结构的层级关系 [98]。动态协作则根据任务进展与智能体状态灵活调整协作方式，如当某个智能体遇到困难时，可请求其他智能体协助。

通信机制 [99] 与共享记忆 [100] 是多智能体协作的基础设施。消息传递协议定义了智能体间的信息交换格式，包括任务请求、进度更新、结果反馈等。共享工作空间允许智能体访问公共数据与中间结果，避免重复计算与信息不一致。冲突解决策略处理智能体间的分歧，例如通过投票机制、优先级规则或引入仲裁者智能体。此外，反思机制可以使智能体能够评估协作效果并调整策略，例如在多轮交互后分析哪些角色分工更合理、哪些通信模式更高效。

3.2.4 部署环境与工程优化

部署环境的选择与优化直接影响智能体系统的响应速度、可靠性与运营成本[101]。

推理加速技术是提升用户体验的关键 [102] 。模型量化通过将浮点参数转换为低比特定点数，在

$$W_q = \text{round}(\frac{W}{s}) \cdot s \tag{47}$$

的映射下将模型大小压缩至原来的25%至50%，同时保持90%以上的性能 [102]。例如，将70B模型从FP16量化至INT4可使其在消费级GPU上运行成为可能。知识蒸馏则通过训练小模型模仿大模型的输出分布 [103]：

$$L = \text{KL}(P_{\text{student}} || P_{\text{teacher}}) \tag{48}$$

在保持大部分能力的同时显著降低推理开销。投机采样 [104]利用小模型快速生成候选token，大模型仅需验证，将延迟降低2至3倍。

缓存策略有效减少重复计算。KV缓存（KV-Cache）存储注意力机制的键值对，使得生成过程中只需计算新token的表示，复杂度从$O(n^2)$降至$O(n)$ [105] 。语义缓存则存储常见查询的完整响应，当检测到相似请求时直接返回缓存结果。Prompt缓存针对共享的系统提示或few-shot示例只计算一次，所有请求复用其表示。这些技术在高并发场景中可将吞吐量提升数倍。

系统可靠性保障需要多层次的工程设计。请求重试机制处理临时性失败，采用指数退避策略

$$t_{\text{retry}} = t_0 \cdot 2^k \tag{49}$$

避免雪崩效应 [106]。而降级策略在主模型不可用时自动切换至备用模型或返回简化响应。此外，负载均衡将请求分配至多个模型实例，避免单点过载。监控告警系统实时追踪延迟、错误率、资源使用等指标，在异常发生时及时介入。日志记录与追踪使得问题排查与性能分析成为可能，特别是在涉及多次工具调用与智能体协作的复杂场景中。

成本优化在大规模部署中至关重要 [107]。批处理推理可以通过同时处理多个请求摊销模型加载与初始化开销。动态批处理根据请求到达情况灵活调整批大小，平衡延迟与吞吐量。模型路由根据请求复杂度选择合适规模的模型，简单任务使用小模型快速响应，复杂任务才调用大模型。混合部署策略结合云端与边缘计算，将低延迟需求的推理放在边缘侧，复杂计算放在云端。这些综合优化措施在保证服务质量的前提下，有望将运营成本降低50%以上 [108]。

4 讨论与开放式问题

4.1 如何系统性地量化与权衡不同评测维度之间的内在冲突?

评测框架中，核心能力与系统工程指标常呈负相关。例如，记忆增强（Memory-augmented）架构通过显式记忆池提升长期上下文一致性，但引入显著的检索延迟与存储开销，损害系统性能 [109]；多智能体（Multi-agent）协作虽能分解复杂任务、提升规划鲁棒性，却增加了通信失败、目标对齐误差等新的失效模式 [16]。当前研究多聚焦于单一维度的优化，缺乏统一的Pareto前沿分析框架，导致“刷榜”行为可能掩盖实际部署中的系统性退化。

LLM Agent的能力涌现本质上是资源消耗与认知复杂度的权衡 [110]。例如，规划与推理的深度（如思维链长度）直接影响首Token延迟（TTFT）；工具调用精度要求多次模型采样，与吞吐量（TPS）矛盾；记忆压缩率与上下文召回率遵循信息论下的率失真权衡。不同维度指标异质且非线性。任务完成度（如GAIA基准的成功率）是离散稀疏信号，而系统效率是连续密集信号，二者难以直接比较。现有工作如AgentBench [111] 虽尝试多维度评分，但仅做线性归一化加权，未能刻画能力边界。

未来的研究可以考虑

- 建立动态效用理论框架：引入基于任务上下文的效用函数，允许不同维度按需动态加权。例如，对实时性要求高的客服场景，延迟权重指数级上升；对科研分析任务，则优先保证推理深

度。

- 设计对抗性协同评测 [112]：构建“压力-响应”测试集，刻意在单一点上施加极端约束（如强制50ms内响应），观察其他能力的迁移与衰退模式，绘制能力-效率弹性曲线。
- 推动硬件-算法联合评测：将模型量化、推测解码等工程优化纳入评测循环，从系统栈视角分析理论能力与物理实现的差距。

4.2 智能体能力的可解释性鸿沟如何阻碍评测有效性，且该鸿沟是否本质上不可弥合？

评测假设“能力可观测、失效可归因”，但LLM Agent的规划、工具选择甚至记忆写入常是涌现行为，缺乏符号化因果链 [50]。例如，ReAct智能体可能在某一步产生正确工具调用，但该决策源于预训练中的隐式模式匹配，而非显式逻辑推导。这导致评测只能判断“对错”，无法定位失效是来自模型知识边界、提示歧义还是架构缺陷。更关键的是，随着模型规模增长，内部表示呈现“过度参数化”特性，可解释性方法（如注意力分析、因果追踪）的解释方差急剧上升。

当前评测多止步于“输入-输出”黑箱评估。即使引入过程监督（如ProcessBench [113]），中间步骤的正确性也无法映射到可干预的模块。例如，Planner–Executor架构中的规划错误，可能源于Planner的幻觉，也可能是Executor反馈信号噪声所致 [114]——二者在端到端评测中混淆。

智能体的强大恰在于其能突破显式编程，但这种突破伴随着因果结构的稀疏化。研究如Rethinking Interpretability [115] 指出，大模型的表示空间存在“语义坍缩”，同一功能可由多个正交子空间实现，导致解释不稳定。这可能暗示完全可解释性可能与最高性能无法兼得：模型把同一任务拆成好几条互不干扰的“暗道”，走任何一条都能到终点，可回头想画地图——每条暗道都长得不一样，解释工具就抓瞎。

未来的潜在方向可能在：

- 发展“能力考古学”评测：不追求实时解释，而是事后通过反事实数据增强（Counterfactual Data Augmentation）构建“能力触发性数据集”，刻画特定能力的激活边界与数据依赖，形成“能力地图”而非“决策路径” [116]。
- 构建可解释性增强的评测协议：在评测中强制插入“解释探针 [117] ”，要求Agent在关键决策点输出结构化解释（如JSON格式的置信度分解），并单独评估解释的自洽性与预测性。若解释与后续行为背离率高，则标记为“伪解释风险”。
- 接受有限可解释性：调优评测目标，从“精确定位故障组件”转向“预测能力失效的统计模式”。例如，通过监控中间表示的异常波动（如基于Gram矩阵的域偏移检测 [118] ）提前预警潜在错误，而非事后归因。

4.3 长期自主运行下，智能体的价值对齐如何随记忆演化而漂移，且现有评测能否捕捉动态对齐失败？

Memory-augmented Agents在长期交互中会持续写入用户反馈、环境信号到新记忆，导致其偏好表示（Preference Representation）发生微漂移 [119]。目录中提到的“长期互动与记忆相关的伦理风险”仅是定性描述。而定量问题在于：当前基于静态红队测试 [120] 或RLHF的对齐评测，假设模型价值观恒定，无法检测渐进式价值侵蚀（Incremental Value Erosion）。例如，一个客服Agent在持续接收用户“越界请求”并获正向反馈后，可能逐步降低拒绝阈值，该过程在单轮评测中完全正常，但在千轮后产生政策违规。

记忆不仅是事实存储，更是奖励信号的累积器。当使用基于检索增强生成（RAG）的记忆时，相似查询会召回带情感极性的历史交互，形成“回音室效应”，强化特定行为模式。现有评测缺乏对记忆写入-检索-强化闭环的长期监控。 几乎所有对齐评测（如HHH Eval [121]）是快照式评估。即便有MT-Bench REF 等多轮对话评测，也仅关注连贯性，而非价值观动态。但价值漂移是非平稳过程，其统计特性随时间改变，从而让传统IID评测 [120] 假设失效。

未来的研究方向可能在：

- 设计“对齐压力测试”协议：模拟对抗性用户，在数千轮交互中持续注入有偏反馈，建立价值漂移检测器。核心指标是“对齐熵增率”——即Agent对同一伦理问题回答的方差随轮次的增长速度。若熵增速率超过阈值，标记为脆弱。
- 构建记忆审计追踪评测：要求Agent维护带版本的记忆日志，评测时回放记忆演化轨迹，检查是否存在“因果倒置”（后续记忆改写前期价值判断的触发条件）。这可将对齐问题转化为可验证的时序一致性检测问题。
- 探索 Constitutional AI 的动态更新：将核心对齐原则置于不可变的“宪法记忆”中，并评测其在长期检索中的可访问性衰减。若宪法记忆的检索权重随新记忆增加而指数下降，则系统存在根本性设计缺陷。此评测需结合知识编辑（Knowledge Editing）[122] 技术，量化核心价值的抗遗忘能力。

5 结论

报告系统梳理了大模型智能体的评测标准与能力影响因素。从评测维度看，智能体的能力评估需同时关注核心智能能力（规划推理、工具使用、记忆管理）与系统工程特性（性能效率、可靠性、安全对齐），两者缺一不可。从影响因素看，智能体能力既取决于大模型的内在属性（架构、训练数据、知识边界），也深受系统设计的制约（提示工程、工具链、多智能体协作、部署优化）。报告总结认为当前评测领域面临三大开放挑战：不同评测维度间存在难以量化的内在冲突与权衡；智能体决策的可解释性鸿沟削弱了评测的有效性与可信度；长期自主运行中的价值对齐漂移尚缺乏动态监测机制。这些问题的解决需要理论突破与工程创新的协同推进，也将决定智能体技术能否真正迈向安全可控的实用化阶段。

6 参考文献

[1] https://www.arxiv.org/pdf/2507.06323

[2] https://arxiv.org/html/2510.09244v1

[3] https://developer.huawei.com/consumer/cn/forum/topic/0202155833118560037

[4] https://arxiv.org/html/2403.16971v5

[5] https://kingy.ai/wp-content/uploads/2025/01/AgentLaboratory.pdf

[6] https://arxiv.org/abs/2511.21382

[7] https://arxiv.org/abs/2210.03629

[8] https://platform.openai.com/docs/guides/function-calling

[9] https://www.emergentmind.com/topics/planner-executor-architecture-4c9e0097-fe2b-4870-b41c-9519c49a07c8

[10] https://arxiv.org/abs/2511.01149

[11] https://www.ema.co/additional-blogs/addition-blogs/build-plan-execute-agents

[12] https://openreview.net/forum?id=WstJ2Wkus3

[13] https://javaaidev.com/docs/agentic-patterns/patterns/orchestrator-workers-workflow/

[14] https://arxiv.org/pdf/2506.03828

[15] https://arxiv.org/abs/1806.02777

[16] https://arxiv.org/abs/2501.06322
[17] https://arxiv.org/abs/2510.20733
[18] https://blog.csdn.net/2401_85375151/article/details/147479954
[19] https://www.themoonlight.io/zh/review/meta-thinking-in-llms-via-multi-agent-reinforcement-learning-a-survey
[20] https://arxiv.org/abs/2507.21504
[21] https://arxiv.org/abs/2509.18661
[22] https://arxiv.org/abs/2508.05668
[23] https://zhuanlan.zhihu.com/p/676368255
[24] https://aws.amazon.com/cn/what-is/reinforcement-learning-from-human-feedback/
[25] https://arxiv.org/abs/2311.01460
[26] https://medium.com/aimonks/chain-of-thought-vs-explicit-chain-of-thought-the-battle-for-ai-reasoning-supremacy-4b057b43ce1c
[27] https://arxiv.org/abs/2508.18870
[28] https://arxiv.org/abs/2508.05366
[29] https://zhuanlan.zhihu.com/p/1926706570470863947
[30] https://zhuanlan.zhihu.com/p/1956140545857552592
[31] https://tridu33.github.io/ZERO-Starting-AI-Planning-in-PDDL-Descripted-World/PlanLanguages/PDDL%E8%AF%AD%E8%A8%80/
[32] https://zhuanlan.zhihu.com/p/1890839022051509427
[33] https://blog.csdn.net/weixin_41496173/article/details/139469231
[34] https://www.themoonlight.io/zh/review/simulating-environments-with-reasoning-models-for-agent-training
[35] https://zhuanlan.zhihu.com/p/690971197
[36] https://developer.volcengine.com/articles/7382246291990347813
[37] https://www.promptingguide.ai/applications/function_calling
[38] https://arxiv.org/html/2508.05298v1
[39] https://medium.com/@sahin.samia/tools-function-calling-in-llms-and-ai-agents-a-hands-on-guide-3a19f2f21954
[40] https://arxiv.org/abs/2510.19208
[41] https://medium.com/intuitively-and-exhaustively-explained/llm-routing-intuitively-and-exhaustively-explained-5b0789fe27aa
[42] https://zhuanlan.zhihu.com/p/1935065663346544823
[43] https://www.sciencedirect.com/science/article/abs/pii/S0926580525002845
[44] https://github.com/nuster1128/LLM_Agent_Memory_Survey
[45] https://arxiv.org/abs/2404.13501
[46] https://aws.amazon.com/cn/blogs/china/agentic-ai-infrastructure-deep-practice-experience-thinking-series-three-best-practices-for-agent-memory-module/
[47] https://medium.com/@vamsikd219/understanding-the-differences-between-llms-rag-and-ai-agents-79778db4bae2
[48] https://arxiv.org/abs/2502.12110
[49] https://www.promptingguide.ai/research/llm-agents
[50] https://www.themoonlight.io/zh/review/llm-based-agents-suffer-from-hallucinations-a-survey-of-taxonomy-methods-and-directions
[51] https://www.themoonlight.io/zh/review/unlocking-the-power-of-multi-agent-llm-for-reasoning-from-lazy-agents-to-deliberation
[52] https://www.cnblogs.com/muzinan110/p/18553196

[53] https://aws.amazon.com/cn/blogs/china/agent-quality-evaluation/
[54] https://qiankunli.github.io/2023/10/30/llm_agent.html
[55] https://www.cnblogs.com/chang09/p/16565815.html
[56] https://www.servicenow.com/docs/bundle/zurich-intelligent-experiences/page/administer/now-assist-ai-agents/concept/ai-agent-dashboard.html
[57] https://medium.com/@raj-srivastava/why-ttft-time-to-first-token-is-the-silent-killer-of-ai-user-experience-2b490c6e991f
[58] https://zhuanlan.zhihu.com/p/1945488445435803643
[59] https://en.wikipedia.org/wiki/Little%27s_law
[60] https://www.zhihu.com/question/1976999445443911873/answer/1977023883245810885
[61] https://www.testwo.com/article/2168
[62] https://medium.com/@georgekar91/measuring-confidence-in-llm-responses-e7df525c283f
[63] https://arxiv.org/html/2404.09127v1
[64] https://zhuanlan.zhihu.com/p/683751917
[65] https://blog.csdn.net/2501_91492197/article/details/155012853
[66] https://www.themoonlight.io/zh/review/on-the-resilience-of-llm-based-multi-agent-collaboration-with-faulty-agents
[67] https://www.themoonlight.io/zh/review/agent-safetybench-evaluating-the-safety-of-llm-agents
[68] https://arxiv.org/html/2505.23815v1
[69] https://blog.csdn.net/YoungOne2333/article/details/147852325
[70] https://zh.wikipedia.org/zh-hans/%E6%AD%90%E7%9B%9F%E4%B8%80%E8%88%AC%E8%B3%87%E6%96%99%E4%BF%9D%E8%AD%B7%E8%A6%8F%E7%AF%84
[71] https://en.wikipedia.org/wiki/Transformer
[72] https://en.wikipedia.org/wiki/Mixture_of_experts
[73] https://en.wikipedia.org/wiki/Neural_scaling_law
[74] https://www.analyticsvidhya.com/blog/2024/09/chinchilla-scaling-law/
[75] https://arxiv.org/abs/2001.08361
[76] https://arxiv.org/html/2411.07715v1
[77] https://qiankunli.github.io/2024/08/17/llm_pre_training.html
[78] https://zhuanlan.zhihu.com/p/16622115263
[79] https://huggingface.co/learn/llm-course/chapter11/1
[80] https://aws.amazon.com/what-is/reinforcement-learning-from-human-feedback/
[81] https://zhuanlan.zhihu.com/p/713737141
[82] https://aws.amazon.com/cn/what-is/retrieval-augmented-generation/
[83] https://aws.amazon.com/cn/blogs/china/sixteen-ways-of-prompt-engineering/
[84] https://www.promptingguide.ai/techniques/cot
[85] https://arxiv.org/abs/2201.11903
[86] https://www.promptingguide.ai/techniques/fewshot
[87] https://developer.volcengine.com/articles/7404769130152198194
[88] https://blog.csdn.net/2401_84495872/article/details/138497127
[89] https://shanechang.com/zh-cn/p/training-llms-smarter-clarifying-ambiguity-assumptions/
[90] https://qianfan.cloud.baidu.com/qianfandev/topic/268600
[91] https://arxiv.org/html/2506.01056v1
[92] https://zhuanlan.zhihu.com/p/712459598
[93] https://www.promptingguide.ai/zh/techniques/react
[94] https://arxiv.org/abs/2510.27287
[95] https://arxiv.org/abs/2402.01680

[96] https://arxiv.org/abs/2510.03463
[97] https://developer.volcengine.com/articles/7451506545462312999
[98] https://zhuanlan.zhihu.com/p/1889789968366408089
[99] https://www.arxiv.org/abs/2510.13821
[100] https://www.53ai.com/news/LargeLanguageModel/2024061090853.html
[101] https://aws.amazon.com/cn/blogs/machine-learning/learn-how-to-build-and-deploy-tool-using-llm-agents-using-aws-sagemaker-jumpstart-foundation-models/
[102] https://zhuanlan.zhihu.com/p/699776257
[103] https://zhuanlan.zhihu.com/p/695640168
[104] https://www.53ai.com/news/finetuning/2024071109285.html
[105] https://huggingface.co/blog/not-lain/kv-caching
[106] https://zh.wikipedia.org/zh-hans/%E9%9B%AA%E5%B4%A9%E6%95%88%E5%BA%94
[107] https://blog.csdn.net/ms44/article/details/141924745
[108] https://aws.amazon.com/cn/blogs/china/accelerate-sagemaker-llm-model-inference-performance-using-rolling-batch/
[109] https://arxiv.org/abs/2502.13843
[110] https://f0jb1v8xcai.feishu.cn/wiki/JPJgwf74ginUjkk0lChcUpRKnGf
[111] https://arxiv.org/abs/2308.03688
[112] https://agijuejin.feishu.cn/wiki/S784w2iDoiwQQpk9bGGc3LW3nQc
[113] https://arxiv.org/abs/2310.08973
[114] https://arxiv.org/abs/2510.07505
[115] https://arxiv.org/abs/2211.06700
[116] https://arxiv.org/html/2504.13897v1
[117] https://zhuanlan.zhihu.com/p/1892988290040329173
[118] https://blog.csdn.net/weixin_44858740/article/details/130049053
[119] https://juejin.cn/post/7562008575533121551
[120] https://zhuanlan.zhihu.com/p/1893610681879726009
[121] https://huggingface.co/datasets/HuggingFaceH4/hhh_alignment
[122] https://arxiv.org/abs/2310.16218