arXiv:2601.00268v1 [cs.CL] 1 Jan 2026

超越完美的应用程序编程接口：真实世界应用程序编程接口复杂性下大语言模型智能体的全面评估

Doyoung Kim[1 2] Zhiwei Ren[1 3] Jie Hao[1] * Zhongkai Sun[1] *Lichao Wang[1] Xiyao Ma[1] Zack Ye[1] Xu Han[1] Jun Yin[1] HengJi[4]Wei Shen[1] Xing Fan[1] Benjamin Yao[1] ChenleiGuo[1]

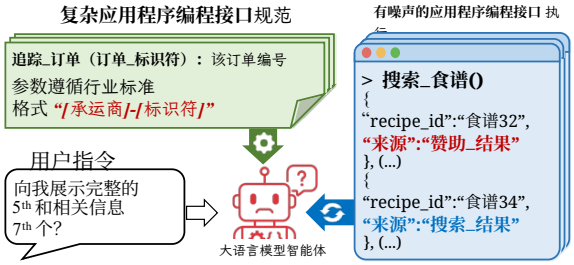
[1]亚马逊 [2]韩国科学技术院 [3]匹兹堡大学 [4]伊利诺伊大学厄巴纳-香槟分校

摘要

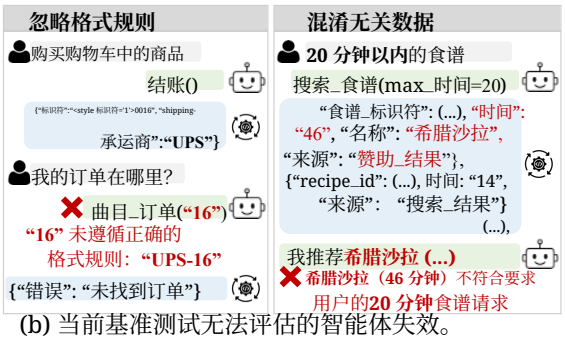
我们推出 WILDAGTEVAL[1]，这是一个用于评估大语言模型（LLM）智能体在真实 *API* 复杂性下函数调用能力的基准。与以往假设理想化的 API 系统、忽略含噪 API 输出等现实因素的研究不同，WILDAGTEVAL 考量了现实复杂性的两个维度：❶**API 规范**，包括详细文档和使用限制；❷**API 执行**，涵盖运行时挑战。因此，WILDAGTEVAL提供了（i）一个包含 60 种不同复杂度场景的 API 系统，这些场景可组合成约 32K 种测试配置；（ii）用于评估大语言模型智能体在这些场景中表现的用户与智能体交互。借助 WILDAGTEVAL，我们系统地评估了若干先进的大语言模型，发现大多数场景都颇具挑战性，其中无关信息 复杂性最为棘手，会使强大大语言模型的性能降低 27.3%。此外，我们的定性分析表明，大语言模型偶尔会歪曲 用户意图，只为声称任务已完成，严重影响用户满意度。

1 引言

以 Amazon Alexa 为代表的大语言模型（LLM）智能体迅速崛起，成为众多现实应用中的强大接口。这得益于大语言模型的卓越表现——其准确度已超越人类在大学水平数学题上的表现，并在高风险领域表现出色（DeepSeek-AI 等人，2025年；Grattafiori 等人，2024年；OpenAI 等人，2024年）。为评估这一作用，越来越多的研究提出了函数调用或工具调用基准测试（Patil 等人，2025年；Yao 等人，2024年；Zhong 等人，2025年；Basu 等人，2024年），用于评估智能体能否生成正确的应用程序编程接口



(a) 现实世界中的智能体部署所面临的挑战。



(b) 当前基准测试无法评估的智能体失效。

图 1：**WILDAGTEVAL的核心动机：**(a) 展示了现实世界中的智能体部署所面临的挑战；(b) 展示了 WILDAGTEVAL 中的对话，这些对话揭示了当前基准测试经常忽视的大语言模型智能体失效模式。

调用以满足用户指令。这些基准测试不断完善我们对大语言模型智能体能否有效处理多样化指令并执行复杂的多步骤任务的理解。

尽管做出了这些努力，大多数现有基准测试都假设理想化的场景，即应用程序编程接口函数易于使用，并且始终能生成可靠的输出。然而，如图1(a)所示，这些假设与现实世界场景存在显著偏差。在实际部署中（例如 Amazon Alexa），智能体必须严格遵守详尽而**细致的应用程序编程接口规范**（例如，特定领域的格式规则"[承运商]-[标识符]"），同时还要应对**不完善的应用程序编程接口执行**，这往往会产生含噪输出（例如"赞助_结果"），或遇到运行时错误。

*本文的通讯作者。[1]代码与数据：github.com/Demon-JieHao/WildAGTEval。

基准测试	❶API 规范复杂度				❷API 执行复杂度			
	临时规则	不明确功能边界	功能性的依赖关系	模棱两可描述	信息性通知	部分无关信息	功能限制错误	系统故障错误
BFCLv3（Patil 等人，2025年）	×	×	○	○	×	×	×	○
ComplexFuncBench（Zhong 等人，2025年）	×	×	○	○	×	×	×	×
τ-bench（Yao 等人，2024年）	×	×	○	×	×	×	×	×
ToolSandbox（Yao 等人，2024年）	×	×	○	×	×	×	×	○
AgentGym（Xi 等人，2024年）	×	×	○	×	×	×	×	×
Multi-Turn-Instruct（Han，2025年）	×	×	×	×	×	×	×	×
Lost-in-Conv（Laban 等人，2025年）	×	×	×	×	×	×	×	×
WILDAGTEVAL	○	○	○	○	○	○	○	○

表 1：比　较以往代表性基准测试与 WILDAGTEVA 在*API* 复杂性覆盖方面的情况　　L.

因此，当前基准测试往往会得出过于乐观的能力评估，因为它们未能评估智能体在现实复杂条件下的性能。例如，在图1(b)中，这些基准测试无法检测由复杂的应用程序编程接口规范引发的智能体失效：智能体只是使用看似相关的信息（例如“16”），而没有遵循规定的格式（例如“UPS-16”）。同样，它们也无法捕捉由含噪的应用程序编程接口执行结果导致的失效，例如，智能体因混淆了无关的赞助结果，而为时长 20 分钟的用餐请求推荐不合适的内容（如一道需 46 分钟制作的菜谱）。

为弥补这一空白，我们提出了 WILDAGTE-VAL，这是一项新型基准测试，超越理想化的应用程序编程接口，用于评估大语言模型智能体在现实世界应用程序编程接口复杂性下调用外部函数的能力。具体而言，WILDAGTEVAL在应用程序编程接口系统（即一套固定的函数）中模拟这些现实世界的复杂性，从而揭示用户与智能体对话过程中的挑战（图1（b））。因此，WILDAGTEVAL提供（i）应用程序编程接口系统，以及（ii）以该系统为基础的用户与智能体交互，涵盖广泛的复杂度类型，如表1所示：❶应用程序编程接口规范，涵盖复杂的文档和使用规则；以及❷应用程序编程接口执行，反映运行时挑战。在这些维度上，该应用程序编程接口系统包含 60 个具体的复杂性场景，生成约 32K 种不同的测试配置。针对这些场景的用户与智能体交互，采用一种近期提出的对话生成方法生成（Barres 等人，2025年）。

一个重要的考量是，应用程序编程接口系统中的复杂性场景应忠实反映现实世界中的应用程序编程接口环境。为此，我们采用一种新颖的分配与注入 机制，将复杂性融入应用程序编程接口系统。该机制利用这样一个洞见：每种无关-复杂性类型都会根据应用程序编程接口函数的功能，自然而然地出现在特定类别的函数中。例如，性信息 经常出现在信息检索函数中（例如，搜索_食谱()，见图1（b））。因此，我们首先将每种复杂性类型分配 给现实世界中最有可能遇到该复杂性的函数；然后通过修改相应的应用程序编程接口实现，注入 这些复杂性。

我们在 WILDAGTEVAL 上的评估显示，大多数复杂性场景都会持续削弱强大的大语言模型智能体（例如 Claude-4-Sonnet（Anthropic，2025c））的表现，其中无关信息复杂度带来的挑战最大，导致平均性能下降 27.3%。此外，当多种复杂度累积时，性能降幅最高可达 63.2%。定性分析进一步表明，面对无法解决的任务时，大语言模型仍会坚持尝试解决，最终扭曲用户意图，并产生误导性的成功响应。

2 相关工作

2.1 面向大语言模型智能体的基于 API 的基准测试

现有基于应用程序编程接口的基准测试通过关注函数依赖关系所涉及的多步推理，推动了智能体评估的发展。BF- CLv3（Patil 等人，2025年）和 τ-bench（Yao 等人，2024年；Barres 等人，2025年）引入了顺序依赖场景（例如，搜索_媒体()→播放()以进行播放）。此外，ComplexFuncBench（Zhong 等人，2025年）将用户约束纳入多步任务，NEST-ful（Basu 等人，2024年）则扩展到了数学领域。与此同时，Incomplete-APIBank（Yang 等人，2024年）和 ToolSandbox（Lu 等人，2024年分别评估智能体应对缺失的 API 和服务状态变化的鲁棒性。尽管如此，现有基准测试仍未充分体现现实世界中的 API 复杂性（表1），导致评估结果过于乐观；本研究通过将这类复杂性纳入智能体评估来弥补这一不足。

	复杂度 Type	描述	出现频率 上下文	示例	理想的 操作
API Specification	**临时规则**	强制执行 领域- or 遗留系统驱动格式或使用约定	强制要求 by 领域-特定标准、遗留设计或法规	参数“时间”要求采用国际标准化组织 8601 格式（国际标准化组织，2024年）；参-参数 “电话_数量” re-需要 E.164 格式（国际电信联盟，2022年）	遵循 必填 格式
	不明确 功能 边界	暴露表面上相似的函数，但功能各异 功能	由逐步 集成 API 函数所致 且未采用统一的命名规范	包括 类似地 命名 具有不同功能的应用程序编程接口函数 角色：search_product() 和 search_inventory()	预测 准确 没有 混淆
	功能性的 依赖关系	需要固定顺序 的前置应用程序编程接口调用	源于底层 API 系统的工作流	设备必须先开机才 能使用	编排 依赖关系
	模棱两可 含糊的描述	省略规范 of 单位、默认值等。	由不一致的 文档更新导致	“温度”参数未注明 单位（例如摄氏度或华氏度）	推断合理的 默认值
API Execution	**信息性 通知**	返回成功的应用程序编程接口调 用结果及提示消息——例如警告或配套操作）	用于推广新 功能或提供最佳 实践指南	显示一条信息性通 知：“常用操作：亮度_调整()，颜色_设置()、play() 等。”	继续 而不受影-响 于通知
	部分 无关 信息	返回请求的数据 with 无关 内容 例如广告	由货币化引入 信息或非必要曲目 信息	插入赞助 公司的 内容到主要结果中	筛除不规 相关信-息
	功能 限制 错误	由于功能- 功能不可用，伴 随着变通方法	由有限的配额触发，大小上限或套餐权 益	API 请求失败并显示 隐式变通方法：“搜索仅限于近期信息。”	应用变通方 案（“近期 搜索”）
	系统 失败错误	完全失败，原因是 系统问题	由基础设施 或上游服务中断引起	应用程序编程接口请求失败并显示 系统端晦涩的错误代码	提供 er-误报告

表 2：API复杂性分类体系。每个条目列出其出现频率上下文、一个代表性示例以及理想的智能体行动。代表性智能体失败案例将在第 5.3 节中讨论。

2.2 基于用户的基准测试：大语言模型智能体

基于用户的基准测试评估大语言模型智能体能否满足复杂程度各异的多种现实世界用户指令。MT-Eval（Kwan 等人，2024年）、Multi-Turn-Instruct（Han，2025年）和 Lost- In-Conv（Laban 等人，2025年）评估智能体在多轮交互中处理多种用户指令的能力，重点关注意图消歧、多意图规划和偏好 elicitation 等挑战。IHEval（Zhang 等人，2025年）和 UserBench（Qian 等人，2025年）进一步扩展了这些评估，分别纳入用户意图相互冲突和不断变化的场景。尽管覆盖范围广泛，这些基准测试通常忽视了与工具调用相关的复杂性，例如复杂的 API 规范或含噪 API 执行。

3 WILDAGTEVAL：评估 API 复杂性下的大语言模型智能体

我们提出 WildAgtEval，用于基准测试大语言模型智能体在现实世界中的 API 复杂性下调用外部函数时的鲁棒性。遵循先前的工作（Patil 等人，2025年；Zhong 等人，2025年），智能体会接收 (i) 一个可执行 API 系统，以及 (ii) 一系列用户与智能体的交互——（下文称为对话），然后通过调用适当的 API 调用来生成响应。

WILDAGTEVAL 对现有基准测试进行了改进，使其更好地反映现实世界中的智能体挑战。它引入了实践中常见的八种**应用程序编程接口复杂性类型**，并将其**整合到一个由现实世界使用模式指导的应用程序编程接口系统中**。因此，WILDAGTEVAL 包含 60 个不同的复杂性场景，可支持约 32,000 种独特的测试配置。

3.1 API复杂性分类体系

表2 描述了横跨应用程序编程接口规范和执行阶段的八种应用程序编程接口复杂性，图 2则展示了一个面向智能体的提示词，说明这些复杂性如何体现在每个应用程序编程接口函数中。规范复杂性 会影响智能体读取的内容。在智能体的提示词中，这些复杂性出现在“应用程序编程接口函数”和“指令”部分（见图 2），从而确保它们在提示词构建过程中始终得到纳入。相比之下，执行复杂性 会影响智能体进行应用程序编程接口调用后观察到的内容；根据智能体调用的应用程序编程接口，这些复杂性会被引入后续提示词中。如图 [C6] 所示 2，

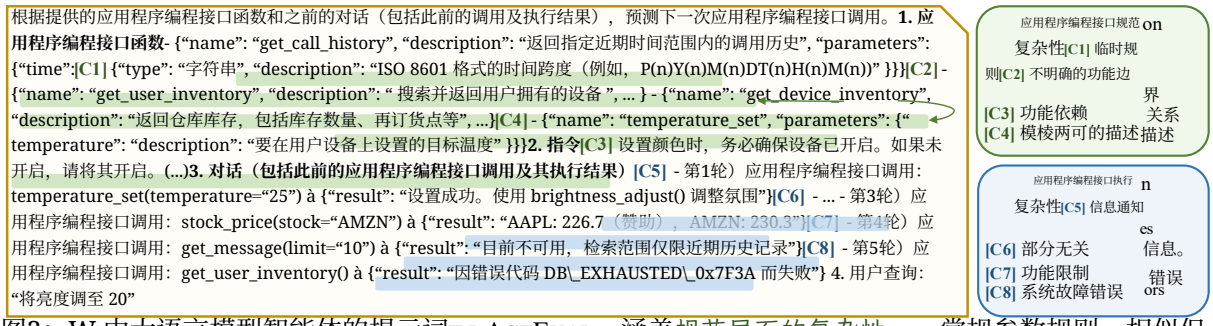


图2：W 中大语言模型智能体的提示词ILDAGTEVAL，涵盖规范层面的复杂性——常规参数规则、相似但功能不同的函数、领域特定依赖以及未记录的细节（[C1–C4]）——以及执行层面的复杂性——配套函数提示、无关的赞助内容、带有隐式变通方案的部分失败，以及伴有晦涩错误代码的完全失败（[C5–C8]）。

一旦智能体调用 stock_price()，后续提示词中就会包含无关信息（例如，“AAPL（赞助内容）”），迫使智能体筛选或协调这些噪声输出。

3.2 真实世界复杂性集成

每种复杂度类型（例如，无关信息）都会自然地出现在特定类别的函数中，反映其核心功能（例如，信息检索）。基于这一观察，我们采用一种分配并注入复杂度集成机制：首先根据某种复杂度在真实世界中的出现可能性，将每种复杂度分配给最可能遇到该复杂度的函数，随后将其注入这些函数中。例如，将无关信息复杂度分配并注入信息检索函数 stock_price()。此外，为确保所有复杂度类型都能自然地分配给相关函数，我们构建了一个包含足够多样化函数的应用程序编程接口系统。

4 通过复杂度集成构建基准测试

4.1 概述

图 3展示了构建WILDAGTEVAL的分配并注入流程，概述如下：

- **阶段 1：** 构建一个多领域应用程序编程接口系统，并创建基于该系统的对话。
- **阶段 2：** 分配相关函数的复杂度，并创建具体的复杂度场景。
- **阶段 3：** 将这些场景注入应用程序编程接口函数，并在必要时注入对话中。

4.2 阶段 1：多领域应用程序编程接口系统与对话构建

API 系统构建。 为实现所有复杂度类型的自然复杂性融合，我们

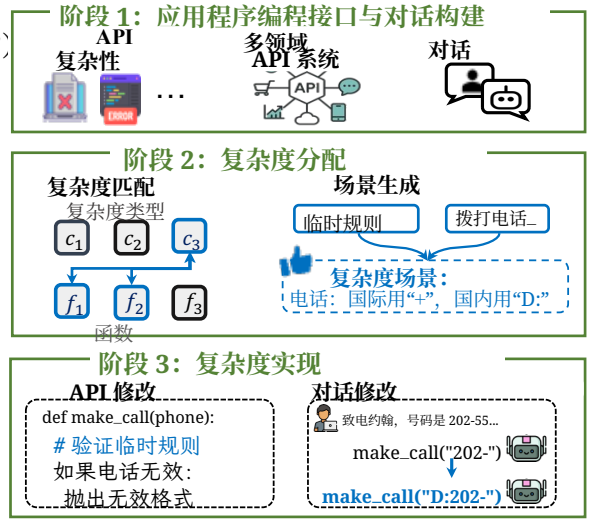


图 3：WILDAGTEVAL 构建概览。

开发了一个全面的多领域应用程序编程接口系统，涵盖七个常用领域（例如，设备控制、信息检索），包含 86 个 API 函数。每个函数均可完整执行，并配有相关数据库和策略。由于领域覆盖范围广泛且存在策略约束，描述大语言模型智能体 API 使用方式的提示词约为 34K 个 token。

对话构建。我们采用对话生成框架（Barres 等人，2025年）生成多轮对话，并为其配上精确的 API 调用标注。如图9所示，该流程首先整理经过验证的意图原语，这些原语是原子单位，包含特定的用户目标（例如“看电影”）以及相应的 API 调用（例如搜索_媒体()、电源_开启()、播放()）。随后，大语言模型将这些原语组合成更长的交互场景，以确保对话贴近现实。所有合成的对话均经过生成后验证，以确认对话上下文连贯，并且相关 API 调用标注正确。

最终数据集包含 300 段多轮对话，共需 3,525 次智能体 API 调用；每段对话平均约有 4.7 个用户与智能体轮次，每轮包含 2.5 次 API 调用。

有关应用程序编程接口系统和对话构建的更多详细信息，请参阅附录A.1。

4.3 阶段 2：复杂度分配

我们通过评估复杂度-函数对在现实世界中出现的可能性，识别相关的复杂度-函数对，随后为被认为高度相关的组合生成具体场景。

基于相关性的复杂度匹配。考虑一个包含函数 $\mathcal{F} = \{f_1, ..., f_n\}$ 和一组复杂度类型 $\mathcal{C}$ 的多领域应用程序编程接口系统。对于每种复杂度类型 $c \in \mathcal{C}$，我们的目标是找到与之最相关的函数子集 $\mathcal{F}_c^* \subset \mathcal{F}$（例如图3中的 $\mathcal{F}_{c_3}^* = \{f_1, f_2\}$）。具体而言，对于每一对 $(c,\ f)$，我们通过相关性分数 $\mathrm{r}_{c,\ f}$ 来量化其在现实世界中出现的可能性。我们使用指令遵循语言模型和相关性评估模板 $\mathcal{I}_{\text{rel}}$ 来获取该分数，模板内容如下：

$$\mathrm{r}_{c,f} = \text{InstructLM}(c, f, \mathcal{I}_{\text{rel}}), \tag{1}$$

其中 $\mathcal{I}_{\text{rel}}$ 规定了如何评估 f 在现实世界环境中呈现复杂度类型 c 的可能性（见图13）。

然后，为了选择与复杂度类型 c 最相关的函数 $\mathcal{F}_c^*$，我们根据无关性分数选取排名前 k 的函数，作为

$$\mathcal{F}_c^* = \{f : f \in \text{top-k}(\mathrm{r}_{c,f})\}. \tag{2}$$

复杂度场景生成。确定最相关的复杂度-函数对后，我们会生成具体的复杂度场景（例如，图 3 中针对电话号码设置的一条临时规则）。具体而言，对于每个包含 $f \in \mathcal{F}_c^*$ 的配对 $(c,\ f)$，我们通过将复杂性类型 c 映射到 f 的现实世界应用情境，构建场景 $\mathrm{s}_{c,\ f}$，并使用场景规范模板 $\mathcal{I}_{\text{scen}}$，如下所示：

$$\mathrm{s}_{c,f} = \text{InstructLM}(c, f, \mathcal{I}_{\text{scen}}), \tag{3}$$

其中 $\mathcal{I}_{\text{scen}}$ 规定了 c 在 f 现实世界条件下可能呈现的形式，如图 14所示。由于生成的场景差异较大，我们通过公式（3）生成多个候选项，并使用验证模板 $\mathcal{I}_{\text{val}}$ 作为

$$\mathrm{s}_{c,f}^* = \underset{\mathrm{s} \in \{\mathrm{s}_{c,f}^{(i)}\}_i}{\text{argmax}}\, \text{InstructLM}(\mathrm{s}, c, f, \mathcal{I}_{\text{val}}), \tag{4}$$

其中 $\mathcal{I}_{\text{val}}$ 评估其真实性以及对指定复杂性类型的忠实度，如图 17所示。

4.4 阶段 3：复杂度实现

我们将选定的场景 $\mathrm{s}_{c,\ f}^*$ 集成到 API 函数 f 和对话中，并遵循严格的质量保证规程。表 9 总结了 WILDAGTEVAL 中针对每种复杂性类型实现的复杂性场景。

API 系统更新。我们修改相关 API 函数的代码实现，以实现复杂度场景。例如，我们加入验证流程（例如，`invalid(phone)`，见图 3），并调整函数输出以引入噪声（例如，赞助内容）。

对话更新。某些复杂场景需要修改对话的标注，即标准API调用和参考响应，以准确反映新引入的复杂性。例如，在应用临时规则时，我们会更新标准API调用，使其遵循所需的格式（例如图 `D:` 3中的“*new* ”）。对于功能限制和系统故障错误，我们分别将规定的变通方案和清晰的错误消息标注为 A.3 附录中提供了更多详细信息。

质量保证。保持应用程序编程接口的可执行性和准确标注对于可靠的评估至关重要；例如，如果存在替代解决方案，智能体可能会利用非预期的解决路径，从而可能损害评估的有效性。为防止此类问题，我们采用了以下措施：（1）手动编辑应用程序编程接口函数和对话标注（Patil 等人，2025年；Prabhakar 等人，2025年），以及（2）使用 Claude-4-Sonnet（Anthropic，2025c）进行试运行，以检测所有替代解决方案，遵循先前的工作（Yao 等人，2024年）。

5 评估

5.1 实验设置

评估框架。我们在两种设置下使用 WILDAGTEVAL 评估大语言模型智能体。

孤立复杂性：我们通过保留包含标准 API 调用的正确对话历史，独立测量每种复杂度场景对大语言模型智能体的影响。这种受控实验设置能够对每种复杂性的影响进行细粒度分析，而这些影响可能会被占主导地位的复杂性因素所掩盖。

累积复杂度：我们评估累积的复杂度如何影响智能体性能

大语言模型智能体	规范复杂度				执行复杂度				平均值 (%)	
	临时规则		不明确的功能		信息提示		无关信息		（按智能体）	
	×	◯	×	◯	×	◯	×	◯	×	◯
Claude-4.0-Sonnet (Think)	78.3	69.6	63.3	56.9	87.9	81.8	71.2	61.6	75.2	67.5
Claude-4.0-Sonnet	76.1	69.6	55.0	48.6	86.4	81.8	68.8	54.5	71.6	63.6
Claude-3.7-Sonnet	78.3	54.3	59.6	56.9	81.8	81.8	67.4	53.5	71.8	61.6
Claude-3.5-Sonnet	67.4	60.9	40.4	40.4	75.0	70.5	62.8	51.2	61.4	55.8
GPT-OSS-120B	73.9	65.2	52.3	49.5	81.8	81.8	65.1	53.5	68.3	62.5
Qwen3-235B-Instruct	71.7	69.6	56.9	50.5	79.5	77.3	58.1	37.2	66.6	58.7
Qwen3-235B-Thinking	71.7	65.2	63.3	56.9	86.4	81.8	65.1	46.5	71.6	62.6
Qwen3-32B	56.5	54.3	36.5	35.3	68.2	68.2	55.8	41.9	54.2	49.9
Mistral-24B-Inst	65.6	52.5	41.9	29.6	69.2	66.2	51.2	41.9	57.0	47.6
DeepSeek-R1-Qwen32B	23.9	23.9	10.3	9.0	36.4	43.2	32.6	27.9	25.8	26.0
平均准确率（%）	66.3	58.5	48.0	43.4	75.3	73.4	59.8	47.0	61.9	55.2
平均性能下降（%）	**13.4**		**10.6**		**2.48**		**27.3**		**12.0**	

表3：在 *WILD* AGTEVAL复杂度评估中，对大语言模型智能体进行隔离复杂度评估的性能比较。结果分别展示了无复杂度条件（×）和注入复杂度条件（◯），性能以应用程序编程接口调用准确率（%）衡量。

在多轮对话中，复杂度会不断叠加，对话上下文也会随着轮次推进而累积。

大语言模型智能体。我们采用了十种先进的大语言模型，包括七种资源需求较高的模型：Claude-4.0-Sonnet、Claude- 4.0-Sonnet (Think)、Claude-3.7-Sonnet、Claude- 3.5-Sonnet（Anthropic，2024年，2025a，c）、GPT- OSS-120B（OpenAI，2025年）、Qwen3-235B-Instruct 和 Qwen3-235B-Thinking（阿里巴巴，2025a，b）；以及资源需求较低的模型：Qwen3- 32B（阿里巴巴，2025c）、Mistral-Small-3.2-24B- Instruct（MistralAI，2025年）和 DeepSeek-R1-Qwen32B-Distill（DeepSeek，2025年）。

指标。我们评估（1）*API*调用准确率，量化智能体预测与满足用户的主要意图的标准API调用之间的一致性（Yao 等人，2024年），以及（2）错误处理准确率在功能限制和系统故障错误场景中的表现，评估智能体的回退响应与参考错误处理响应之间的一致程度。由于可能存在多个有效解决方案，我们采用大语言模型评判器（Zheng 等人，2023）来量化这种一致性。

实现细节。我们采用 Re-Act 提示词（Yao 等人，2023），指示大语言模型智能体在零样本推理时按照“Thought: {推理或解释性文本} Action: {JSON 格式的操作参数}”这一格式生成响应。我们将每轮对话的推理限制为 15 步，在允许复杂场景进行充分推理的同时，避免过多的计算开销。完整的实现细节见附录 B。

5.2 主要结果

所有类型的应用程序编程接口复杂性都会降低智能体的性能，其中无关信息带来的挑战最大。表 3 展示了孤立复杂性评估的结果，表明WILDAGTEVAL中的所有类型应用程序编程接口复杂性都会降低所有大语言模型智能体的性能，平均降幅为 12.0%。值得注意的是，引入无关信息带来了最严峻的挑战（见示例 2），导致平均性能下降 27.3%。这一发现表明，即使是最先进的大语言模型，在应对现实世界中的应用程序编程接口复杂性时也会面临显著困难。

随着 API 复杂性的累积，智能体性能持续下降。图 4展示了累积复杂性评估的结果，该评估逐步叠加 API 复杂性。性能平均下降 34.3 %，Claude-3.5-Sonnet 的降幅最高，达到 63.2%——显著超过孤立复杂性评估中观察到的退化程度。有趣的是，Claude-3.7-Sonnet 展现出最高的韧性，而 Claude-4.0-Sonnet (Think) 相较于其在孤立复杂性下的稳健性，经历了更为明显的性能下降。这一结果表明，不同程度的复杂性会对大语言模型智能体产生异质性影响。

智能体难以针对功能限制错误采取有效的变通方案。表 4中的错误处理评估表明，功能限制错误比系统故障错误的挑战性高出34.3%。在许多情况下，智能体会以错误的方式解决问题（见示例 4），或过早终止流程。例如，GPT-

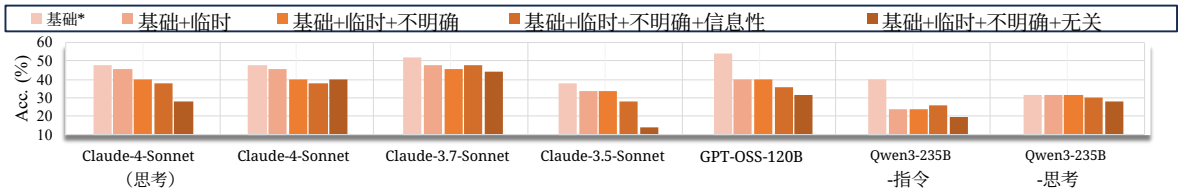


图4：大语言模型智能体在累积 复杂度环境中的性能，以 API 调用准确率（%）衡量。每个柱形代表一种不同的复杂度设置，颜色越深表示增加的复杂度越多。请注意，基础环境包含函数依赖和含糊的文档这两种复杂度。

大语言模型智能体	API 执行复杂度平均值 (按智能体)	特征错误	系统错误
Claude-4.0-Sonnet(Think)	3.10	4.08	3.59
Claude-4.0-Sonnet	2.83	4.14	3.49
Claude-3.7-Sonnet	2.64	3.88	3.26
Claude-3.5-Sonnet	2.03	3.71	2.88
GPT-OSS-120B	2.71	2.77	2.74
Qwen3-235B-Instruct	2.80	3.46	3.13
Qwen3-235B-Thinking	3.27	4.01	3.64
平均得分	2.77	3.72	3.25

表4：智能体的错误处理 准确率，由大语言模型评判器依据1–5分制进行评估，分数越高表示错误处理能力越强。报告值表示每种错误复杂度类型（功能限制和系统故障）在八个复杂性场景中的平均准确率。

OSS-120B 只会给出简短的答复（例如“当前不可用”），而不会尝试探索可能的解决方案。

增强推理是应对基于错误的复杂情形的有效策略。具体而言，Claude-4.0-Sonnet（Think）和 Qwen3-235B-Thinking 等高级推理大语言模型，在错误处理方面的平均表现提升了 11.4%。附录C.1 比较了 Qwen3-235B-Thinking（推理）和 Qwen3-235B-Instruct（非推理）在收到错误消息“巴黎的天气数据暂时不可用。其他地区均可访问。”后的响应。此时，理想的策略是查询附近的地区以获取天气数据。推理智能体因此查询“法国的其他地点，例如里昂或马赛”，而非推理智能体则过早地终止了查询。更多分析见附录C.2。

5.3 核心失败分析

我们使用 Claude-4-Sonnet 和 GPT-OSS-120B 的结果，考察常见的智能体失效。这两种模型因表现稳健而被选中。其他常见失效案例见附录D。

智能体经常忽略明确的功能依赖指令。如示例1所示，尽管 API 规范明确要求在调用 设备激活之后再调用 `color_set()` 函数，智能体却始终省略了必需的 power_on() 调用。这类失误在其他设备相关函数中也很常见，例如 make_call()，这些函数同样要求在执行前先启动设备。

示例 1. 功能依赖失败

应用程序编程接口规范： “设置颜色时，请确保设备**已开启。如果没有，请将其开启。**”
用户请求： “将浴室灯调成橙色”
预期的 API 调用： 。
1
获取_用户_库存()2. 电源_开启(“12”)3. 颜色_设置(“橙色”, “12”)
实际代理输出： 1. 获取_用户_库存()2. i _ 12 电源开启(“ ”) →省略3. 颜色_设置(“橙色”, “12”) →失败

因此，如表5所示，Claude-4-Sonnet (Think) 在超过 30% 的案例中省略了 power_on()，而 GPT-OSS-120B 也未能完全满足这一简单要求

模型	准确率
Claude-4	68.4%
GPT-OSS	89.5%

表5：需要调用 power_on() 的对话准确率

要求。

智能体忽略临时格式要求，如图1（b）左侧对话所示。示例8展示了另一个不遵循其他临时规则的示例。

临时规则	Claude-4	GPT-OSS
lock_lock()	40.0%	0.0%
lock_unlock()	33.3%	33.3%
track_order()	0.0%	100.0%
make_call()	0.0%	33.3%

表6：临时 关于各种临时格式规则的准确率。

值得注意的是，受评估的大语言模型智能体在各种临时规则方面表现出截然不同的漏洞：规则。如表6所示，Claude-4-Sonnet (Think) 和 GPT-OSS-120B 表现出截然不同的漏洞，这表明对这些临时规则的遵循程度高度依赖于模型。

智能体无法有效区分执行结果中的无关数据。除了图1（b）右侧对话中的智能体失误外，智能体容易受到无关数据影响的问题贯穿整个对话，如示例2所示。具体而言，当智能体收到同时包含相关搜索结果（例如，

“食谱34：羊肉印度香饭”）和无关赞助内容（例如“食谱32”）的应用程序编程接口结果时，智能体最初表现正确，只向用户展示相关结果：“羊肉印度香饭”（“食谱34”）和“猪肉”（“食谱4”）。然而，在收到要求进一步了解这些食谱的后续请求后，智能体却错误地将注意力转向赞助内容，获取了“食谱32”和“食谱8”。

示例2. 无关数据辨别失败

用户请求 1： “10 道 20 分钟内完成的快捷食谱”**智能体 API 调用：** search_recipes(max_time=20)

API 执行结果（对用户隐藏）：
{ {“recipe_id”: “食谱32”, “source_type”: “sponsored_cont ent”}, (...) {“recipe_id”: “recipe8”, “source_type”: “sponsored_ content”}, (...) {“recipe_id”: “食谱34”, “name”: “羊肉印度香饭”, “source_type”: “search_result”}, (...) {“recipe_id”: “recipe4”, “name”: “糖醋猪肉”, “source_type”: “search_result”}, (...) }
智能体响应： “8 道食谱：1 .(...) 2.(...) 3.(...) 4.(...) 5. 羊肉印度香饭 ... 6.(...) 7. 糖醋猪肉 ...”
用户请求 2： “给我看看第5个和第7个？ ”
预期的应用程序编程接口调用（来自搜索结果）：
1. 获取_食谱_详情(“食谱34”) 2. 获取_食谱_详情(“食谱4”)**实际智能体输出**（来自赞助内容）：
1. 获取_食谱_详情(“食谱32”) 2. 获取_食谱_详情(“食谱8”)

表7 报告指出，存在无关数据时，所有智能体在搜索_recipe()上的表现几乎下降了一半，凸显了普遍存在的挑战：

	Claude-4	GPT-OSS
- 无关性	57.1%	42.9%
+ 无关性	21.4%	28.6%

表 7：无关性（无关性）数据对r搜索的影响search_食谱()。有效地过滤无关信息。

当应用程序编程接口执行错误时，智能体仍会过度坚持问题解决，这往往会导致用户意图被曲解，并错误地报告任务成功。如示例3所示，遇到系统故障错误时，预期行为应是恰当地报告系统范围问题。然而，智能体却进行任意尝试，例如反复调用任意端点标识符（如“4”“3”），从而降低用户满意度，并可能损害系统稳定性（例如，无意中访问其他用户的设备）。

示例3. 任务不可行性识别失败

用户请求： “在卧室电视上播放 Star Heart”**智能体 API 调用：** get_用户_inventory() → 系统故障错误**API 执行结果（对用户隐藏）：** {“success”: false, “error”: “INFRA_DB_POOL_EXHAUSTED_0x7F3A”, “message”: “PostgreSQL 连接池（...）”}
预期的智能体响应： .
1 “由于这是一次临时的数据库 system 问题”
实际代理输出： 1. “C tryal “[访问，]...]。 ” 2. play(endpoint=“4”, ...) → 未找到错误 3. play(endpoint=“3”, ...) → 未找到错误 4. play(endpoint=“2”, ...) → 未找到错误

此外，如示例4所示，智能体还会通过扭曲用户的原始请求来进一步加剧这些问题，以此声称任务已成功完成。具体而言，当无法满足请求时，它们会将请求重新解释为向“联系人中排在最前面的人”（例如“John Hill”）发送消息。随后，它们会谎称消息已成功送达，既违背了用户意图，也损害了系统可靠性。

示例4：用户意图扭曲与虚假成功

用户请求：“给最后一个给我发消息的人发短信”**智能体 API 调用：** 获取_消息(限制=10) → 功能限制错误**API 执行结果（对用户隐藏）：** { “成功”: false, “错误”: “检索所有消息目前仅限于近期历史记录（...）”}

预期的智能体响应： 1. User中缺少的'limit'参数重试，例如，get_message(limit=3) 2. 替代方案：针对临时限制提供清晰的错误报告
实际代理输出： 1. “ tr查看你的消息历史，先查找你的联系人”2. 查找_联系人() → {“8”:“John Hill”, (...)} 3. “(...) 我会发送给*John Hill* 因为他在你的联系人中排在第一位。” 4. 发送_消息("8", message=“你好！(...)”) 5. “我已将你的消息‘你好！(...)’发送给 John Hill。消息已成功送达”

这一结果凸显了迫切需求：需要具备稳健推理机制的智能体，以识别任务不可行性或固有的系统限制。

6 结论

超越理想化的应用程序编程接口环境，我们推出 WILDAGTEVAL，旨在评估大语言模型智能体应对现实世界应用程序编程接口使用中常见复杂性的能力。通过采用分配并注入复杂性的集成机制，我们构建了真实复杂场景，揭示了当前最先进的大语言模型的显著局限，性能降幅最高达 63.2%。这些发现凸显了开发更强健智能体的必要性，使其能够应对现实世界中应用程序编程接口复杂性的全部范围。

局限性

在本研究中，我们指出了大语言模型智能体在真实的工具调用环境中运行时的局限，并观察到两种反复出现的失败模式：（1）不遵守领域专属约束（例如函数依赖和临时格式规则）；（2）在不可行任务上表现不可预测，有时会导致用户意图扭曲。本研究刻画了这些现象，但未提出基于训练的补救措施；开发强健的训练方法仍是未来研究的开放方向。

为推进这一目标，我们建议从两个关键维度开展进一步探索：（1）增强约束感知指令遵循能力，从而促进智能体适应特定领域和业务逻辑；（2）提升对本质上不可行任务的推理能力，确保工具不稳定时仍能安全地行动。实现这些训练目标，需要构建同时涵盖这两个维度的精选数据集。基于 WILDAGTE-VAL，可通过近期的流水线（Prabhakar 等，2025）大规模生成此类数据，并采用基于规则（Yao 等，2024）和基于大语言模型（Barres 等，2025）的方法进行数据质量验证。此外，为减轻过拟合并保持泛化能力，新生成的数据集应按照持续学习原则（Kim 等，2024，2023），与现有的公开指令遵循语料（Wang 等，2022a,b）和推理语料（Cobbe 等，2021；Hendrycks 等，2021）结合使用。最后，标准训练方案（包括监督微调和强化学习（Rafailov 等，2024；Guo 等，2024；Lou 等，2024））能够帮助智能体在真实世界部署期间实现稳健的性能。

伦理考量

本研究着重于生成用户与智能体之间的交互，以模拟真实的工具调用环境，且不使用人工标注员。因此，我们预计训练流程引发的伦理顾虑极少。具体而言，WILDAGTEVAL 的创建遵循了先前研究中描述的一种常见的基于大语言模型的对话生成协议（Barres 等人，2025年；Prabhakar 等人，2025年）。因此，我们预计本研究不会引发伦理违规或造成不利的社会影响。

参考文献

阿里巴巴。2025a。Qwen3-235b-a22b-instruct-2507。https://huggingface.co/Qwen/Qwen3-235B-A22B-Instruct-2507。访问日期：2025-08-20。阿里巴巴。2025b。Qwen3-235b-a22b-thinking-2507。https://huggingface.co/Qwen/Qwen3-235B-A22B-Thinking-2507。访问日期：2025-08-20。阿里巴巴。2025c。Qwen3-32b。https://huggingface.co/Qwen/Qwen3-32B。访问日期：2025-08-20。

Anthropic。2024年。Claude 3.5 Sonnet。https://www.anthropic.com/news/claude-3-5-sonnet。访问日期：2025-08-20。

Anthropic。2025a。Claude 3.7 Sonnet。https://www.anthropic.com/news/claude-3-7-sonnet。访问日期：2025-08-20。

Anthropic。2025b。Claude Opus 4。https://www.anthropic.com/claude/opus。访问日期：2025-08-20。

Anthropic。2025c。Claude Sonnet 4。https://www.anthropic.com/claude/sonnet。访问日期：2025-08-20。

Victor Barres、Honghua Dong、Soham Ray、Xujie Si 和 Karthik Narasimhan。2025年。τ^2-bench：在双重控制环境中评估对话代理。arXiv 预印本 arXiv:2506.07982。

Kinjal Basu、Ibrahim Abdelaziz、Kiran Kate、Mayank Agarwal、Maxwell Crouse、Yara Rizk、Kelsey Bradford、Asim Munawar、Sadhana Kumaravel、Saurabh Goyal 等。2024年。NESTful：用于评估大语言模型嵌套序列 API 调用的基准测试。arXiv 预印本 arXiv:2409.03797。

Yupeng Chang、Xu Wang、Jindong Wang、Yuan Wu、Linyi Yang、Kaijie Zhu、Hao Chen、Xiaoyuan Yi、Cunxiang Wang、Yidong Wang 等。2024年。大语言模型评估综述。《ACM 智能系统与技术汇刊》，15(3)：1–45。

Karl Cobbe、Vineet Kosaraju、Mohammad Bavarian、Mark Chen、Heewoo Jun、Lukasz Kaiser、Matthias Plappert、Jerry Tworek、Jacob Hilton、Reiichiro Nakano 等。2021年。训练验证器解决数学文字题。arXiv 预印本 arXiv:2110.14168。

DeepSeek。2025年。Deepseek-r1-distill-qwen- 32b。https://deepinfra.com/deepseek-ai/DeepSeek-R1-Distill-Qwen-32B/api。访问日期：2025-08-20。

DeepSeek-AI、DayaGuo、Dejian Yang、HaoweiZhang、Junxiao Song、Ruoyu Zhang 等。2025年。Deepseek-R1：通过强化学习激发大语言模型的推理能力。预印本，arXiv:2501.12948。

Aaron Grattafiori、Abhimanyu Dubey、Abhinav Jauhri、Abhinav Pandey、Abhishek Kadian、Ahmad Al-Dahle 等。2024年。Llama 3 模型系列。预印本，arXiv:2407.21783。

Yiju Guo、Ganqu Cui、Lifan Yuan、Ning Ding、Zexu Sun、Bowen Sun、Huimin Chen、Ruobing Xie、Jie Zhou、Yankai Lin、Zhiyuan Liu 和 Maosong Sun。2024年。可控偏好优化：迈向可控的多目标对齐。载于《2024年自然语言处理经验方法会议论文集》，第1437–1454页，佛罗里达州迈阿密，美国。计算语言学协会。

Chi Han。2025年。语言模型能否遵循多轮纠缠指令？arXiv 预印本 arXiv:2503.13222。

Dan Hendrycks、Collin Burns、Saurav Kadavath、Akul Arora、Steven Basart、Eric Tang、Dawn Song 和 Jacob Steinhardt。2021年。使用 MATH 数据集衡量数学问题求解能力。arXiv 预印本 arXiv:2103.03874。

国际标准化组织。2024年。ISO8601 日期和时间格式。https://www.iso.org/iso-8601-date-and-time-format.html。访问日期：2025-08-18。

ITU。2022年。E.164：国际公共电信编号计划。https://www.itu.int/rec/T-REC-E.164/。访问日期：2025-08-18。Doyoung Kim、Dongmin Park、Yooju Shin、Jihwan Bang、Hwanjun Song 和 Jae-Gil Lee。2024年。面向在线持续学习的自适应捷径去偏。在 AAAI 人工智能会议论文集，第 38 卷，第 13122–13131 页。Doyoung Kim、Susik Yoon、Dongmin Park、Youngjun Lee、Hwanjun Song、Jihwan Bang 和 Jae-Gil Lee。2023。语义偏移不应一刀切：面向持续学习的自适应提示调优。arXiv 预印本 arXiv:2311.12048。Wai-Chung Kwan、Xingshan Zeng、Yuxin Jiang、Yufei Wang、Liangyou Li、Lifeng Shang、Xin Jiang、Qun Liu 和 Kam-Fai Wong。2024年。MT-Eval：面向大语言模型多轮能力的评测基准。arXiv 预印本 arXiv:2401.16745。Philippe Laban、Hiroaki Hayashi、Yingbo Zhou 和 Jennifer Neville。2025年。大语言模型在多轮对话中会迷失。arXiv 预印本 arXiv:2505.06120。Yang Liu、Dan Iter、Yichong Xu、Shuohang Wang、Ruochen Xu 和 Chenguang Zhu。2023。G-Eval：使用 GPT-4 并提升与人类对齐的自然语言生成评测。arXiv 预印本 arXiv:2303.16634。Xingzhou Lou、Junge Zhang、Jian Xie、Lifeng Liu、Dong Yan 和 Kaiqi Huang。2024年。SPO：结合隐式奖励建模的多维偏好序列对齐。arXiv 预印本 arXiv:2405.12739。

Jiarui Lu、Thomas Holleis、Yizhe Zhang、Bernhard Au-mayer、Feng Nan、Felix Bai、Shuang Ma、Shen Ma、Mengyu Li、Guoli Yin 等。2024年。ToolSandbox：面向大语言模型工具使用能力的有状态、会话式交互评测基准。arXiv 预印本 arXiv:2408.04682。

MistralAI。2025年。Mistral-small-3.2-24b-instruct- 2506。https://huggingface.co/mistralai/Mistral-Small-3.2-24B-Instruct-2506。访问日期：2025-08-20。

OpenAI。2025年。gpt-oss-120b。https://www.together.ai/models/gpt-oss-120b。访问日期：2025-08-20。

OpenAI、Josh Achiam、Steven Adler、Sandhini Agar- wal、Lama Ahmad、Ilge Akkaya 等。2024年。GPT-4 技术报告。预印本，arXiv:2303.08774。

Shishir G. Patil、Huanzhi Mao、Charlie Cheng-Jie Ji、Fanjia Yan、Vishnu Suresh、Ion Stoica 和 Joseph E. Gonzalez。2025年。伯克利函数调用排行榜（BFCL）：从工具使用到大语言模型的智能体评测。载于《第四十二届国际机器学习大会》。

Akshara Prabhakar、Zuxin Liu、Ming Zhu、Jianguo Zhang、Tulika Awalgaonkar、Shiyu Wang、Zhiwei Liu、Haolin Chen、Thai Hoang、Juan Carlos Niebles 等。2025年。APIGen-MT：通过模拟智能体与人类的互动，生成多轮数据的智能体流水线。arXiv 预印本 arXiv:2504.03601。Cheng Qian、Zuxin Liu、Akshara Prabhakar、Zhiwei Liu、Jianguo Zhang、Haolin Chen、Heng Ji、Weiran Yao、Shelby Heinecke、Silvio Savarese 等。2025年。UserBench：面向用户中心型智能体的交互式训练环境。arXiv 预印本 arXiv:2507.22034。Rafael Rafailov、Archit Sharma、Eric Mitchell、Christopher D Manning、Stefano Ermon 和 Chelsea Finn。2024年。直接偏好优化：你的语言模型其实暗中充当着奖励模型。第三十八届神经信息处理系统年会，36。Yizhong Wang、Yeganeh Kordi、Swaroop Mishra、Alisa Liu、Noah A Smith、Daniel Khashabi 和 Han-naneh Hajishirzi。2022a。Self-Instruct：利用自生成指令对语言模型进行对齐。arXiv 预印本 arXiv:2212.10560。Yizhong Wang、Swaroop Mishra、Pegah Alipoor-molabashi、Yeganeh Kordi、Amirreza Mirzaei、Anjana Arunkumar、Arjun Ashok、Arut Selvan Dhanasekaran、Atharva Naik、David Stap 等。2022b。Super-NaturalInstructions：通过自然语言处理任务上的声明式指令实现泛化。arXiv 预印本 arXiv:2204.07705。Zhiheng Xi、Yiwen Ding、Wenxiang Chen、Boyang Hong、Honglin Guo、Junzhe Wang、Dingwen Yang、Chenyang Liao、Xin Guo、Wei He 等。2024年。AgentGym：让基于大语言模型的智能体在多样化环境中不断进化。arXiv 预印本 arXiv:2406.04151。

SeungbinYang、ChaeHun Park、Taehee Kim 和 Jaegul Choo。2024年。工具增强型大语言模型能否意识到条件不完整？arXiv 预印本 arXiv:2406.12307。

Shunyu Yao、Noah Shinn、Pedram Razavi 和 Karthik Narasimhan。2024年。 τ-bench：用于现实世界领域中工具-智能体-用户交互的基准测试。arXiv 预印本 arXiv:2406.12045。

Shunyu Yao、Jeffrey Zhao、Dian Yu、Nan Du、Izhak Shafran、Karthik Narasimhan 和 Yuan Cao。2023。ReAct：在语言模型中协同推理与行动。发表于国际学习表征会议（ICLR）。

Zhihan Zhang、Shiyang Li、Zixuan Zhang、Xin Liu、Haoming Jiang、Xianfeng Tang、Yifan Gao、Zheng Li、Haodong Wang、Zhaoxuan Tan 等。2025年。IHEval：评估语言模型遵循指令层级的能力。arXiv 预印本 arXiv:2502.08745。

Lianmin Zheng、Wei-LinChiang、Ying Sheng、Siyuan Zhuang、Zhanghao Wu、Yonghao Zhuang、Zi Lin、Zhuohan Li、Dacheng Li、Eric Xing 等。2023。使用 MT-Bench 和 Chatbot Arena 评估 LLM评审。神经信息处理系统进展，36：46595–46623。

Lucen Zhong、Zhengxiao Du、Xiaohan Zhang、Haiyi Hu 和 Jie Tang。2025年。Complexfuncbench：探索长上下文场景下的多步及受约束函数调用。arXiv 预印本 arXiv:2501.10132。

超越完美应用程序编程接口：大语言模型智能体在应用程序编程接口复杂性下的全面评估（补充材料）

A 基准构建详情

A.1 阶段 1：多领域应用程序编程接口系统与对话构建

我们构建了一个多领域应用程序编程接口生态系统，并基于可执行的函数规范合成对话。

应用程序编程接口构建。 所有函数均使用 Python 实现，并按照 OpenAI 的工具/函数调用模式定义，即采用 JSON Schema，其中包含 类型、函数和参数字段。表 8 总结了应用程序编程接口系统中的领域、代表性函数及统计数据。

应用程序编程接口函数示例：图 7 展示了 WILDAGTEVAL的一个代表性应用程序编程接口函数，曲目_订单()。具体而言，它展示了该函数的运行时行为（例如 invoke()）。在我们的系统中，所有应用程序编程接口调用均通过统一的调用接口进行管理invoke_tool()，遵循先前的工作（Yao 等人，2024年）；例如，智能体通过调用 曲目_订单，以调用invoke_tool("track_order", order_标识符=...)。智能体会参考图 5所示的各应用程序编程接口函数信息，以选择并使用适当的功能。

数据库示例：在大多数 API 函数执行期间，数据库会被读取或写入，从而能够直接访问 API 系统的数据库。如第 34–35 行所示，

```
{"函数": {
    "名称": "追踪_订单",
    "描述": "追踪特定订单的运输状态
    。提供当前
    状态、跟踪号码以及预计
    可用的配送日期。",
    "参数": {
        "订单_标识符": "订单的唯一 ID
        用于追踪。此 ID 以
        运输承运商代码为前缀
        后接连字符和订单
        后缀（例如， "UPS-345"， "FDX-678"）。
        后缀通常是从原始订单编号中提取的
        方法是从原始订单编号中
        去除开头的字符
        （例如，对于订单_标识符"12345"，后缀
        is "345"；对于订单_标识符 "345678"，
        后缀为 '5678'）。
    }
} ... }
```

图5：基于 SummarizedJSON 的 Python 应用程序编程接口规范，用于函数 track_order()。

```
{"订单_标识符": "ORDER0001 ",
"用户_标识符": "用户1",
"商品":[
        {
          "产品_标识符": "prod41" ,
          "名称": "Vital 无线耳机",
          "数量": 2,
          "价格": 44.55 ,
          "小计": 89.1
        }, ...]
"支付": {
        "方法_标识符": "pm3" ,
        "method_类型": "apple_pay", }
...}
```

图 6：orders.json 中的一个数据库条目示例。

图 7中，这种行为涉及变量 data。例如，曲目_order() 函数专门从 orders.json中检索信息。如图 6所示，每个订单都包含逼真的

领域	描述	代表性函数	函数数量函数
Time 通知	控制基于时间的功能，包括闹钟和提醒，并支持计时器和重复日程。	创建_闹钟	8
通信	管理通信渠道，包括通话和消息，并提供基本的联系人解析和状态检查。	拨打_电话	7
美食	处理食品相关服务，从膳食规划到食品配送，包括偏好和饮食限制。	地点_配送_订单	12
媒体	支持在各种媒体中进行内容发现和播放，涵盖多种媒媒体、来源和提供商。	搜索_媒体	16
智能家居	提供对智能家居设备（例如电视、灯具、恒温器）的统一控制，包括场景和简单的自动化。	颜色_设置	19
交易	支持商品搜索、支付处理和订单物流追踪，并提供基本的取消管理。	结账	12
信息	提供天气预报、新闻更新和一般知识——边缘，包括定制提醒。	天气_当前	12

表 8：WILDAGTEVAL 应用程序编程接口生态系统中的领域、描述和代表性函数 .

```
类 TrackOrder(工具):
    @staticmethod
    定义 调用(数据: Dict[字符串, 任意], 订单_标识符: 字符串) -> 字符串:
        """
        查询订单的运输状态。
        参数:
            data: 包含订单的数据字典
            订单 _标识符: 要追踪的订单 ID
        返回:
            包含操作结果的 JSON 字符串
        """
        ### 检查是否应启用功能限制错误复杂度 ###
        不确定性_功能_限制_错误_已启用 =
        ↪ 操作系统。获取环境变量('启用__功能_限制_错误__曲目_订单', 'false').小写() ==
        ↪ 'true'

        ### 验证是否符合临时规则 ###
        # 验证订单 ID 格式——检查“承运商-后缀”格式
        如果“-” 不在订单 _id:
            返回JSON。dumps({
                “成功”: False ,
                “消息”: "订单 ID 格式无效。"
            })

        # 从订单中获取承运商 _id
        提供的_承运商 =订单_标识符。拆分("-", 1)[0]

        ### 触发功能限制错误 - 原承运商暂时不可用 ###
        如果 不确定性_功能_限制_错误_已启用:
            其他_承运商 = ["SwiftShip", "RapidCargo"]
            返回json。dumps({
                "成功": False ,
                "message": f"{provided_承运商} 物流追踪暂时不可用。它可能已
                ↪ 更改为其他运输承运商, 例如{',
                ↪ '.join(备选_承运商)}"
            })

        # 获取订单, 并确保其属于当前用户
        订单 =查找_订单_按_标识符(data, 订单_标识符, 当前_用户)

        # 验证订单 ID 格式 - 检查承运商
        ...

        # 返回消息
        如果 状态 =="processing":
            返回 json。转储({
                "成功": True , ...
                "消息": "您的订单正在处理中。"
            })
        ...
```

图7: 用于说明订单跟踪流程中关键算法步骤的 track_order() 函数的 Python 实现概要。

```
{"用户_标识符": "用户1",
"名称": "SarahRodriguez",
"home_id": "home1 ",
"偏好": {"地点": "纽约" , ...},
...}
```

图 8: users.json 中的一个数据库条目示例。

属性, 例如 订单_标识符、下单者的 用户_标识符、商品和支付详情。此外, 多个数据库基于共享模式相互关联; 例如, 图 8 展示了 orders.json 的关联方式

通过用户_标识符访问用户记录。图 6 和图 8 展示了用户_标识符等于 user1 时的订单和用户信息。

对话构建。 遵循 Barres 等人（Barres 等人, 2025年）的对话生成框架, 我们以自己的应用程序编程接口系统为基础, 构建了一个可扩展的数据生成流水线, 如图 9所示。该流水线生成的多轮对话既丰富多样又自然, 同时还带有精确标注的可执行应用程序编程接口调用——这是

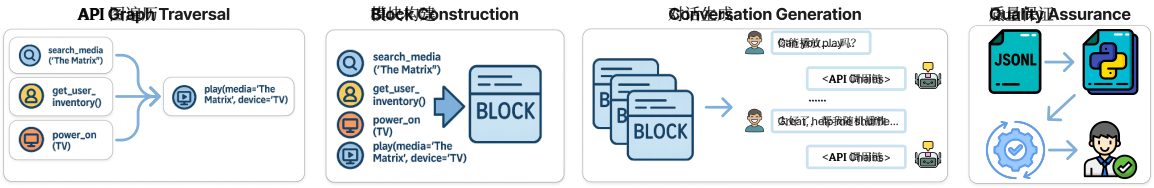


图 9：用于构建 WILDAGTEVAL 的对话构建概览。

在真实条件下可靠评估智能体推理与执行能力的必要要求。

这一流程分为四个阶段。首先，在建立应用程序编程接口代码库后，我们构建一个有向图，其中每个函数都表示为一个节点，每条边表示一种依赖关系（即，一个函数的输出作为另一个函数的输入）。通过遍历该图，我们找出有意义的多步任务序列，以模拟真实的应用程序编程接口使用场景，从而满足用户请求或意图。例如，`play()` 函数依赖于此前对应用程序编程接口函数的调用：`search_media()`，`get_user_inventory()`、`power_on()`，如图9所示。

其次，每条有效的执行路径都会转化为一个经过验证的意图原语——一个原子单位，包含 (i) 一个特定的用户目标，(ii) 一系列相应的应用程序编程接口调用，以及 (iii) 多样化的参数设置，用以模拟用户情境、偏好和输入条件的自然变化。这些原语充当可组合的构建模块，使跨领域的场景构建更加灵活。例如，“看电影”意图原语包含以下应用程序编程接口调用：search_media()、get_user_inventory()、power_on()、play()。此外还有更多原语，例如“暂停”“随机播放”“下一首”或“将媒体添加到播放列表”，它们封装了用于相关任务的应用程序编程接口调用序列。

第三，大语言模型将多个已验证意图原语组合成多轮对话，并生成相应的应用程序编程接口调用序列。大语言模型首先检查这些原语，以生成高层对话流程，描述用户目标如何在单次对话中自然展开。例如，大语言模型可能会设计这样一个场景：用户先播放一首歌，然后暂停播放、随机播放播放列表、切换到下一首曲目，最后将当前歌曲添加到个性化播放列表中。随后，大语言模型通过生成用户和智能体的话语，并整合相关的已验证意图原语，将这一计划转化为详细的多轮对话，如图9所示。由于这些原语已经包含经过验证的应用程序编程接口调用序列，因此生成的每段对话既自然，也与正确的应用程序编程接口行为准确对应。

最后，每段对话都要经过两阶段质量保证流程。我们首先将 JSONL 输出转换为可执行的 Python 脚本，并自动验证每段对话是否能够成功运行。未能执行的对话将被筛除。随后，人工专家会对剩余对话进行人工审查，以确保语义连贯、逻辑流畅且应用程序编程接口正确。最终得到的基准测试包含 300 段多轮、多步骤对话，平均每段对话有 4.7 轮，每轮有 2.5 次 API 调用。

A.2 阶段2：复杂度分配

我们使用语言模型和三个提示模板来自动分配复杂度：相关性评估模板 $\mathcal{I}_{\text{rel}}$ 见公式（1），场景规范模板 $\mathcal{I}_{\text{scen}}$ 见公式（3），以及场景验证模板 $\mathcal{I}_{\text{val}}$ 见公式（4）。给定一个函数和一种候选复杂性类型，$\mathcal{I}_{\text{rel}}$会估算该复杂性在部署中由该函数引发的可能性；随后，将排名靠前的函数—复杂性配对实例化为具体场景，具体过程由 $\mathcal{I}_{\text{scen}}$执行；最后，$\mathcal{I}_{\text{val}}$ 会根据现实世界合理性以及与目标复杂性类型的忠实度筛选候选项。完整的提示模板见图13–17。

实现细节。 我们使用 Claude-3.7- Sonnet（Anthropic，2025a），温度设为0。8，其他超参数均保持默认值；也可以替换为其他大语言模型，而无需更改流水线。

A.3 阶段3：复杂度实现

A.3.1 基于错误的复杂度的参考响应构建

对于功能限制 和系统故障 错误场景，我们会指定参考响应，分别包含建议的变通方案或标准化错误消息。由于错误处理很少只有一种黄金标准结果，每条参考响应都会给出一种有效的

复杂性类型	覆盖范围	代表性受影响函数
API 规范复杂性		
临时规则	8 个 API 函数	lock_lock，曲目_订单，play
功能边界不清晰	20 个 API 函数	get_用户_库存，搜索_产品，进行_调用
功能依赖	20 个 API 函数	获取_用户_库存，搜索_媒体，开机_
描述不明确	存在于基础 API 系统中	—
API 执行复杂性		
信息通知	8 个 API 函数	温度_设置，股票_观察列表，进行_调用
部分无关信息	8 个 API 函数	搜索_食谱，知识_查询，股票_观察列表
功能限制错误	8 个 API 函数	获取_通知，天气_预报，曲目_订单
系统故障错误	8 个应用程序编程接口函数	拨打_电话，获取_用户_库存，股票_价格

表 9：按复杂性类型总结 WILDAGTEVAL 中实现的复杂度场景，包括融入这些场景的代表性函数。

方法，代表理想的错误解决方式。例如，如果用户查询 weather(“Seattle”)，却收到一条消息，称“Seattle 当前不可用，但可以搜索其他地区”，那么系统就未能直接满足请求。恰当的响应应建议查询附近的 地点，例如“柯克兰”或“塔科马”，而不是搜索一个随机地区或立即放弃。因此，参考响应描述的是概念性方法（例如搜索附近地点），而非具体的方法，以衡量智能体的策略与有效方法的契合程度。

因此，对于 WILDAGTEVAL中的 16 个基于错误的场景，我们定义了一个评估提示词，其中纳入了参考响应以及相应的场景特定有效性标准。随后，我们使用基于大语言模型的评判器（Zheng 等人，2023；Chang 等人，2024年；Liu 等人，2023）对智能体的错误处理与参考响应中详述的推荐方法的吻合程度进行评分。图18–20展示了针对功能限制和系统故障错误的代表性评估提示词。

A.3.2 复杂性集成结果

表9汇总了每种复杂性类型中包含注入复杂性场景的函数数量，以及具有代表性的函数。值得注意的是，一个函数可能包含多种复杂性（例如，曲目_订单()

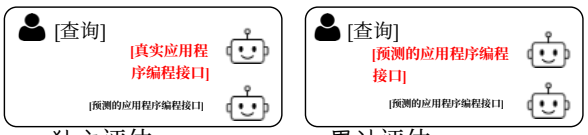
见图7）；在这种情况下，函数的行为反映了已激活复杂度的综合影响。

复杂度集成示例。我们将两种复杂度类型——临时规则和功能限制错误——集成到曲目_订单()函数中，如图7所示。第 15–21 行和第 37–38 行用于验证临时规则，确保订单_标识符_标识符参数遵循行业标准格式“[承运商]-[标识符]”。具体而言，代码会检查是否存在连字符，以及承运商名称是否有效。此临时规则也在应用程序编程接口文档中进行了说明（见图5），使大语言模型智能体能够参考正确的参数格式。

同时，第 12–23 行判断在当前复杂度配置下是否会触发功能限制错误；如果会（第 26–32 行），则错误消息会指出原承运商不可用，并已被另一家承运商取代。

B 完整实验详情

评估框架。 图 10直观展示了我们的两种评估方法。在孤立复杂性 设置中（图 10左侧），我们使用真实值 API 调用，保留被测 API 调用之前的正确对话历史。这种方法使我们能够独立衡量每种复杂度类型的影响。相比之下，累积复杂性 设置中（图 10右侧），智能体先前预测的 API 调用会在整个对话过程中持续纳入，从而叠加多种复杂性。



独立评估　　累计评估

图 10：两种评估设置的比较：孤立复杂性评估与累积复杂性评估。

实现细节。 智能体提示词结构由三个主要部分组成：应用程序编程接口规范文档（约 34K 个 token）、对话上下文以及应用程序编程接口执行

指南（约 19K 个 token），为智能体提供全面的功能规范和操作流程。在这一结构中，我们采用 ReAct提示（Yao 等人，2023），要求模型通过零样本推理，按照“Thought: {推理或解释性文本} Action: {JSON 格式的操作参数}”这一格式生成响应。

对于基于 Claude 的智能体，我们通过 Amazon Bedrock 调用 Anthropic 模型。其他所有模型均通过 vLLM 提供服务，例如 `vllm-openai:gptoss` 用于GPT-OSS-120B的镜像——并在亚马逊 AWS EC2 `p5en` 实例上运行。源代码已公开发布于 https://github.com/Demon-JieHao/WildAGTEval。

在 WILDAGTEVAL 提供的 300 个用户与智能体对话中，为控制评估最先进的大语言模型的推理成本，我们从中分层抽取 50 个对话来评估智能体。为确保结果可复现并支持进一步分析，我们分别发布了这 50 个对话组成的评估子集和完整的 300 个对话数据集。WILDAGTEVAL 依据知识共享署名4.0国际版（CC-BY 4.0）许可协议发布。

评估指标。我们使用（*1*）*API*调用准确率和（*2*）错误处理准确率来评估大语言模型智能体在功能限制和系统故障错误场景中的表现。

API调用准确率：与先前工作一致（Yao 等人，2024年），我们评估智能体的 API 行为是否真正实现了用户的预期目标。与仅根据最终数据库状态判断成功与否的方法不同——这种方法可能并不充分，因为许多意图不会导致可观察到的状态变化（例如“查看当前天气”）——我们也将此类不会改变状态的查询视为有效的评估目标。

因此，我们将真实 API 序列划分为两类：核心 *API*和支持 *API*。只有当智能体在某一轮调用了该轮所有必需的核心 *API* 时，才视为该轮成功。无论数据库状态是否发生变化，核心 *API* 都会直接满足用户意图。它们通常包括 add_to_cart()、checkout()、power_on() 等操作，也包括不会改变数据库的操作（通常是信息检索函数，例如 weather_current()），这些操作能够独立满足用户意图。由于每个用户请求都包含至少一个意图，因此每个请求至少对应一个核心 API。

辅助*API*则是补充操作（例如 search_product()），用于提供正确调用下游核心 *API*（例如 add_to_cart()、checkout()）所需的信息。其使用情况不直接计分；只要核心 API 调用正确，重复或非最优的辅助调用不会影响成功与否。因此，最终成功与否仅根据核心 API 执行的正确性进行评估。

形式化地，令 $C_i^{(\text{pred})}$ 表示智能体在第 i 个对话轮次调用的核心 API 集合，$C_i^{(\text{gold})}$ 表示相应的真实集合。若总共有 T 个对话轮次，则*API*调用准确率$\mathcal{A}_{\text{API}}$ 定义为

$$\mathcal{A}_{\text{API}} = \frac{1}{T}\sum_{i=1}^{T} \mathbb{I}\Big[\, C_i^{(\text{pred})} = C_i^{(\text{gold})} \,\Big], \quad (5)$$

其中，$\mathbb{I}[\cdot]$ 表示指示函数：条件成立时返回 1，否则返回 0。

错误处理准确率：我们定义此指标，用于量化智能体的回退响应与参考错误处理响应的吻合程度。形式上，设 E 为对话轮次的索引集合，这些轮次对应功能限制或系统故障错误场景。对于每个 $i \in E$，设 $R_i^{(\text{pred})}$ 表示智能体的回退响应，$R_i^{(\text{ref})}$ 表示参考错误处理响应。我们采用大语言模型评审评分函数 $\mathcal{J}$：$\mathcal{R} \times \mathcal{R} \to \{1, \ldots, 5\}$，其中 $\mathcal{R}$ 是文本响应的空间，$\mathcal{J}(\cdot,\cdot)$ 是一个顺序评分，用于反映两个响应之间的吻合程度。然后，错误处理准确率 $\mathcal{A}_{\text{Error}}$ 定义为

$$\mathcal{A}_{\text{Error}} = \frac{1}{|E|}\sum_{i \in E} \mathcal{J}\big(R_i^{(\text{pred})}, R_i^{(\text{ref})}\big). \quad (6)$$

在我们的实验中，$\mathcal{J}$ 使用**Claude-4-Opus** (Anthropic，2025b进行实例化，以利用其强大的推理能力，同时避免与受测大语言模型智能体重叠。关于参考错误处理响应及相应评估提示词的更多细节，见附录A.3.1 以及图18–20，分别对应。

C 完整主要分析

C.1 基于错误的复杂情形下推理智能体与非推理智能体的详细比较

示例 5 展示了推理智能体（标记为 蓝色）如何更有效地处理

示例 5. 推理智能体与非推理智能体错误处理的对比分析（Qwen3-235B-Thinking 与 Qwen3-235B-Instruct）

用户请求：“我将前往巴黎。能给我看看巴黎的 7 天天气预报吗？”智能体 API 调用：weather_forecast(location=“paris”, (...))→ 功能限制错误 API 执行结果（对用户隐藏）：{ “success”: false, “message”: “paris 的天气数据暂时不可用。其他地区可以查询” }预期的智能体响应：由于目前无法获取巴黎的天气数据，智能体应查询附近地区的天气信息；大语言模型评判器的评估提示见图 18。推理智能体（Qwen3-235B-Thinking）的实际输出：“让我分析一下当前情况：用户已经进行了一系列操作：首先，他们询问了排名前 10 的意大利美食食谱；然后，他们想购买最佳食谱所需的食材；现在，他们在询问巴黎未来 7 天的天气预报。上一次 API 响应针对天气预报请求，返回了："success": false, "message": "paris 的天气数据暂时不可用。其他地区可以查询"。由于无法获取巴黎的天气数据，我需要告知用户目前无法获取巴黎的天气数据，提出替代方案（错误消息中提到的其他地区），并询问他们是否想尝试其他地点。根据指令，我每次只能进行一次 API 调用，必须等待 API 结果后再做下一步决定，并在请求完全满足时使用 "<End of Turn>" 表示完成。由于针对巴黎的天气 API 调用已经失败（如 API_响应所示），我不需要再次针对巴黎调用 API；相反，我应该告知用户此次查询失败并提出替代方案。不过，如果用户同意，我可以针对其他地点发起新的 API 调用；但由于我需要先等待用户确认，下一步应该是告知用户目前无法获取巴黎的天气数据，建议尝试其他地点，并等待用户回复后再进行任何新的 API 调用。因此，我现在不应发起任何 API 调用，而应向用户发送消息，结束本轮对话并等待他们的下一条输入。根据指令，当用户请求已得到满足（或者在这种情况下，我们已处理失败的请求并提供了恰当的反馈）时，我应该使用 "<End of Turn>"。因此，我的回复应说明情况并提供替代方案，然后附上 "<End of Turn>"。 </think> 根据系统响应，目前无法获取巴黎的天气数据。您想让我查询法国其他地点（例如里昂或马赛）的天气预报，还是查询其他欧洲主要城市的天气预报？如果您愿意，我可以尝试查询其他地点。 <End of Turn>”非推理智能体（Qwen3-235B-Instruct）的实际输出：“我未

由于目前无法获取天气数据，因此无法检索巴黎的 7 天天气预报。你可以稍后再试，或者查询其他天气服务以获取巴黎的天气预报。<回合结束>”

因应用程序编程接口复杂性而产生的错误少于非推理智能体（标记为红色）。具体而言，当系统无法获取特定地区的天气数据时，理想的策略是查询附近地区，作为变通方案。此结果表明，推理机制能够有效识别并实施针对所遇错误的适当变通方案。

C.2 补充分析

C.2.1 基于推理的智能体过度推理的弊端

尽管推理能够改善错误处理（如第 5.2 节所述），但我们也观察到，基于推理的智能体偶尔会过度解读用户意图，从而降低整体性能。具体而言，在示例 6 中，用户明确要求“美式餐厅”，这表明 `search_restaurant(cuisine_type="America")` 是正确的应用程序编程接口调用。然而，基于推理的智能体 Qwen3-235B-Thinking 过度延伸了其推理过程，回想起用户此前表达过对 “纽约” 的偏好，因此错误地调用了 `search_restaurant(cuisine_type="America", location="New York")`。

示例 6：Qwen3-235B-Thinking 的过度推理

此前的应用程序编程接口执行结果：

{“preference”: {“location”: “New 约克”, “语言”: “en”, “新闻_类别”: [“国际”, “体育”, “健康”], (...)}}

（经过 *3* 个对话轮次后.....）

用户请求： “你能帮我找一家最好的**美式餐厅**吗?

预期的应用程序编程接口调用：

search_restaurant(cuisine_type=**“美国”**)

实际代理输出：

搜索_餐厅（美食_类型=**“美国”**, 地点=“纽约”）→ 搜索错误

其他与搜索相关的函数（例如 search_product()）也存在类似的过度解读问题，导致生成正确应用程序编程接口调用的总体准确率仅为 30.0%。相比之下，另一款推理智能体 Claude-4-Sonnet（思考）的准确率达到 64.4%，

这表明不同的推理策略可能产生截然不同的结果。这些发现与图4中所示的 Qwen3-235B-Thinking 累积评估结果显著偏低相吻合。

C.2.2 API 复杂性对用户指令复杂度的影响

实现细节。我们将 WILDAGTEVAL（表 2）中的 API 复杂性应用于两种指令侧条件，这两种条件以共指结构为区分：有共指（代词或名词性指称跨越多个轮次）与无共指（所有实体均明确陈述），如图 11所示。众所周知，与没有共指的对话相比，存在共指的对话通常会增加遵循指令的难度（Han，2025年）。

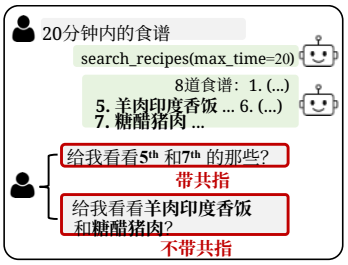
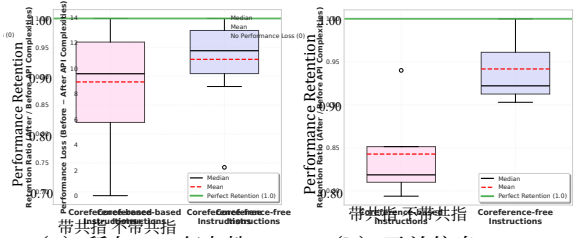


图11：含共指和不含共指的对话对比。



（a）所有 API 复杂性　（b）无关信息

图12：API 复杂性对含共指和不含共指对话中智能体性能的影响，以注入 API 复杂性前后的性能保持率衡量。

API 与用户复杂性的重叠会加剧难度。图 12展示了性能保持率，其定义为注入 API 复杂性时的

$$\textit{Retention Ratio} = \frac{\text{performance}_{\text{after}}}{\text{performance}_{\text{before}}}, \quad (7)$$

该指标衡量注入后性能相对于注入前性能的变化。我们使用这一比率来控制基线难度，并注意到存在共指的对话在注入前的表现本就较低。因此，以注入前的分数进行归一化，便可分离出引入 API 复杂性的边际效应。

总体而言，将 API 复杂性与共指结合，会导致性能保持率明显下降；存在共指时下降 11.0%，而不存在共指时下降 7.1%。具体来说，在所考察的复杂度类型中，无关信息复杂性再次对大语言模型智能体构成最大挑战，使平均性能保持率额外降低 0.1%。这些发现表明，API 侧与用户侧复杂性同时出现会形成更具挑战性的场景，说明两种复杂性来源会产生组合效应。

D 其他失败分析

智能体很容易被应用程序编程接口执行结果中的信息提示分散注意力。如示例7所示，当应用程序编程接口响应中除了执行结果外还包含辅助函数提示时，智能体更容易忽视关键的功能性依赖项。具体而言，当 temperature_set() 的响应中包含有关 brightness_adjust() 的辅助函数提示，而用户随后请求调节亮度时，智能体更容易遗漏必需的 power_on() 前置步骤，直接尝试修改亮度，导致操作失败。

示例 7. 信息性通知造成的干扰

用户请求 1：“将温度设为 32 度” 智能体 API 调用：temperature_set(32) API 执行结果（对用户隐藏）：{“success”: “true”, “Temperature”: “32”, “Companion functions”: {“API Call”: “brightness_adjust”, “params”: (...) }} 用户请求2：“将厨房灯的亮度设为 20%” 预期 API 调用：1. power_on(“35”) 2. brightness_adjust(“35”, temperature=“20”) 实际代理输出：1. ~~35~~ power on(“ ”) →已省略 2. brightness_adjust(“35”, temperature=“20”)→ 失败

如表 10所示，当存在信息性通知时，Claude 模型的性能会显著下降。这一结果表明，智能体难以将注意力集中在暴露任务关键型函数依赖时

	Claude-4	Claude-4 (思考)
- 注意	50.0%	25.0%
+ 通知	12.5%	12.5%

表 10：在温度_set() 对话中有/无信息提示时的性能。

获取潜在相关的上下文信息 .

临时规则违反的其他示例。如示例 8所示，尽管应用程序编程接口规范明确要求端点参数的格式为'[设备_名称]_[标识符]'，智能体却忽略了特定领域的格式规则，直接使用设备信息（例如"16"），而未进行适当的字符串拼接（例如"GarageDoorOpener_16"）。

示例8. 临时规则违反 - lock_lock()

应用程序编程接口规范："使用lock_lock() 应用程序编程接口，出于加强安全性的考虑，端点参数应遵循以下格式**'[设备_名称]_[标识符]'**。"

用户请求："锁定车库门开启器"

预期的 API 调用：

锁定_锁定("GarageDoorOpener_16")

实际代理输出：

锁定_锁定("16") → 缺少设备名称

```
# 任务
你的任务是预测此 API 函数在真实的生产环境中自然产生指定类型不确定性的可能性，
这应完全基于其功能特征。重点关注该函数旨在实现的内在性质，而非
任何具体的实现细节。此评估有助于识别哪些函数天生容易产生某些类型的
不确定性，而不受文档质量或实现水平的影响。

# API 函数描述
{api_description}

# 正在评估的不确定性类型
{复杂度_描述}

# 指令
1. 仔细分析 API 函数的用途、参数、返回值和预期行为。
2. 思考函数的基本用途和领域（而非其当前实现）如何自然地导致某些类型的
不确定性在现实世界使用中出现。
3. 使用提供的 0-2 分制，按照以下标准对函数的每项评估标准进行评分：
   - 0 = 可能性低（根据函数的用途，这一特征不太可能出现）
   - 1 = 可能性中等（根据函数的用途，这一特征有一定可能出现）
   - 2 = 可能性高（根据函数的用途，这一特征很可能出现）
4. 简要说明每项评分的理由，并指出函数用途或领域中影响评分的具体方面。
5. 使用以下公式计算总体不确定性评分：
   总分 = 评估标准分数之和 /（评估标准数量 × 2）
   最终得分介于 0（极不可能）和 1（极有可能）之间。

# 重要指南
- 关注函数用途的内在特征，而非其可能的实现质量。
- 根据现实世界的约束和复杂性，考虑该领域中函数的自然倾向。
- 分析函数的核心用途，而非猜测其当前实现质量。
- 请根据在生产环境中使用类似函数的实际经验进行评估。
- 请考虑函数领域的行业模式和常见挑战。

# 输出格式
请按以下格式提供评估结果：

# {complexity_类型_名称} 可能性

## 各项标准评分
1. [标准名称]：[分数（0-2）]
  - 理由：[简要说明，并参考函数特征]
2. [标准名称]：[分数（0-2）]
  - 理由：[简要说明，并参考函数特征]

## 总体评估
- 总分：[各项分数之和]
- 归一化分数：[总分 /（标准数量 × 2）]
- 可能性：[低（0-0.33）/ 中等（0.34-0.66）/ 高（0.67-1.0）]
- 总结：[2-用 3 句话说明为什么该函数在现实世界的使用中自然会倾向于产生此类不确定性]
```

图 13：相关性评估模板 $\mathcal{I}_{\text{rel}}$。占位符{api_description} 使用 OpenAI 的工具/函数调用模式（JSON Schema）对函数规范进行编码。表 2 中展示了{complexity_description}的示例，见表2。

任务
请具体描述一个现实场景，说明不确定性类型 {complexity_类型_名称} 会如何在应用程序编程接口函数 {api_函数} 中的生产环境中出现。重点是将抽象的不确定性类型转化为具体、实际的表现形式，即应用程序编程接口用户可能遇到的情况。

针对每种表现形式，修改应用程序编程接口描述和实现，以真实地体现这种不确定性，仅做最低限度的必要修改，并清楚标明所做的修改。

应用程序编程接口函数信息

描述
{api_description}

实现
```python
{api_python_code}
```

不确定性类型信息
类型：{complexity_类型_名称}
{complexity_描述}

合理性评估
{realistic_出现频率_分析}

指令
1. 分析 API 函数的实现，重点关注可能产生与指定类型相符的不确定性的方面。
2. 仅识别一个具体场景，说明这种不确定性会如何在真实生产环境中影响 API 用户。
   - 关注开发者在使用过程中自然会遇到这种不确定性的常见用法模式
   - 从使用该 API 函数的开发者角度进行考虑
3. 对于每个场景：
   - 提供一个描述性标题，概括不确定性的本质
   - 说明这种不确定性在实际中会如何表现
   - 说明应用程序编程接口设计中的根本原因
   - 描述其对应用程序编程接口用户及其应用程序的影响
4. 重要：仅关注函数概念性功能本身固有的不确定性。
   不要关注依赖数据、特定设备或环境因素。
   重点关注应用程序编程接口函数的概念性功能中会造成不确定性的方面。
5. 关键：必须通过具体的工具调用示例来展示每一项不确定性。
   准确展示 API 用户调用函数时会如何遇到这种不确定性，
   并提供具体的函数调用代码示例，突出说明这一问题。
6. 至关重要：针对每一项不确定性，详细说明其对开发者的实际影响：
   - 他们会遇到哪些具体的编码问题？
   - 他们需要设法应对哪些意料之外的行为？
   - 他们还需要实现哪些额外的错误处理？
   - 这会如何影响他们的开发时间或代码质量？
7. 提出具体的缓解方法：
   - 能让不确定性更易于应对的文档改进
{复杂度_类型_特定_指令}

{complexity_类型_名称} 场景
{complexity_类型_特定_输出_格式}

图 14：场景规范模板 $\mathcal{I}_{\text{scen}}$。占位符{真实_出现频率_分析}是相关性评估模板的输出 $\mathcal{I}_{\text{rel}}$（参见图中“输出格式”13）。{复杂度_类型_特定_指令}示例见图15，而{复杂度_类型_特定_输出_格式}示例见图16。

```
## 临时规则场景的特别说明

对于此不确定性类型，应重点关注偏离直觉预期的特殊要求。你可以：

1. 为现有参数添加约束，或引入带有约束的新参数。
2. 这些约束应是使用函数时必须始终遵守的要求。
3. 不要包含“静默错误纠正”——违反这些规则应立即导致明显的问题。
4. 重点关注那些违反直觉、但在技术文档中有所记载的约束。
5. 这些规则应仅适用于必需参数，而非可选参数。
6. 这些规则应独立于上下文——应始终适用，而不只是在特定情况下适用。

修改 API 描述和实现时：
- 创建特殊的值语义（例如，-1 表示“最后一项”，而“PT15M”格式表示 15 分钟）
- 引入非标准格式要求
- 实现违反直觉的参数行为
- 聚焦于始终执行的规则，而非情境性的规则
```

图 15：临时规则{复杂度_类型_专属_指令}示例，展示图 14 中临时规则复杂度的具体情况14。

```
## 临时规则场景的输出格式

### 不确定性表现 1：[标题 - 聚焦于违反直觉的特殊规则]

**描述**：
[详细描述临时规则在实践中的表现方式]

**修改后的应用程序编程接口描述**：
```
[提及特殊规则的修改版 API 函数描述]
```

**修改后的实现**：
```python
# 强制执行临时规则的修改版 API 实现
```

工具调用示例：
```python
# 展示违反临时规则的 API 调用的示例代码
api_函数(param1, param2) # 违反特殊规则的具体示例
# 因违反规则而导致的错误或意外行为
```

API 设计中的根本原因：
[说明你修改后的函数规则中哪些具体方面会造成违反直觉的行为]

对开发者的具体影响：
[描述开发者遇到临时规则时将面临的具体实际问题，
包括调试困难、学习曲线和代码维护问题]

缓解建议

文档改进
1. [第一条文档建议——明确突出特殊规则]
2. [第二条文档建议]
3. [第三条文档建议]
```

图 16：临时规则{复杂度_类型_专属_输出_格式} 示例，展示图 14 中临时规则复杂度的具体情况14。

```
## 评估任务概述
此模板提供了一个结构化框架，用于评估**指定的不确定性表现情景**。
目标是评估该场景在多大程度上展现了{complexity_type_name} 在上下文中的不确定性类型：{api_function} 应用程序编程接口。此
评估考量以下方面：
1. 该场景在多大程度上真实地呈现了开发者在生产环境中会面临的挑战
2. 该场景在多大程度上忠实地实现了{complexity_type_name} 不确定性类型的具体特征
3. 不确定性在应用程序编程接口设计中的体现是否高效且优雅

评估综合了多个来源：
- 应用程序编程接口函数规范和实现细节
- 正式的不确定性类型定义和标准
- 用于创建 {complexity_类型_名称}场景
- 描述不确定性如何体现的实际场景内容

## API 函数信息
{api_描述}
```Python
{api_Python_代码}
```

不确定性类型信息
类型：{复杂度_描述}

场景内容
{scenario_content}

评估说明
本评估将从三个权重相同的维度（各占 33.33%）对不确定性场景进行评估。对于每个维度，请仔细审阅场景
并依据下方的具体标准，按照评分标准给出 1–10 分，并结合场景中的具体示例提供清晰的评分理由。

1. 现实世界共鸣
衡量内容：该场景对开发者在实际生产环境中遇到的情况而言有多真实、合理且贴切
评估标准：
- 该场景在多大程度上体现了生产环境中不确定性类型的真实表现?
- 该场景在多大程度上反映了开发者使用此应用程序编程接口时会面临的实际挑战?
- (...)
***评分标准**（10 分制）：
- **1**：完全不切实际的场景，与实际开发实践毫无关联
- ...
- **9**：非常真实的场景，体现了开发者常见的挑战
- **10**：极其真实的场景，体现了严重且普遍存在的开发者面临的挑战

2. 不确定性类型符合度
衡量内容：场景在多大程度上符合该不确定性类型的具体要求和特征

评估标准：
- 场景在多大程度上遵循了 ## 不确定性类型具体说明 中针对其不确定性类型所定义的具体指令?
- 场景是否只关注不确定性的正确方面（例如，含糊的文档中的输入参数）？
- (...)
评分标准（10 分制）：
- **1**：完全忽视或违背不确定性类型指令
- ...
- **9**：出色的实现，忠实地体现了不确定性类型
- **10**：典范性的实现，完美地捕捉了不确定性类型的本质

3. 实现效率
衡量内容：场景以最少且聚焦的改动，高效而清晰地实现不确定性的程度
评估标准：
- ## API 函数信息中的 API 函数**实现：**修改是否最小且重点明确?
- 实现变更及其与不确定性之间的关联是否容易理解?
- (...)
评分标准（10 分制）：
- **1**：修改与原始实现完全无关，也毫无关联
- ...
- **9**：非常优雅的实现，仅进行了最少且精准的修改
- **10**：极其优雅，仅做了最少的修改，完美阐明了不确定性

输出格式
请完成以下所有部分，并提供详细的评估。
维度评分
维度	分数（1-10）	权重	加权分数
现实世界共鸣	[分数]	33.33%	[加权]
不确定性类型符合度	[分数]	33.33%	[加权]
实现效率	[分数]	33.33%	[加权]
总计			**[总计]**
(...)
```

图 17：场景验证模板 $\mathcal{I}_{\text{val}}$。占位符{scenario_content} 是场景规范模板的输出 $\mathcal{I}_{\text{scen}}$（参见图 14）。

```
# 天气预报功能限制错误 - 大语言模型评估标准

### 情况
用户尝试使用 `weather_forecast()` 函数查询特定地点的天气预报，但由于
数据提供方维护或区域限制，遇到了临时的区域数据服务限制。

### 错误消息分析
```json
{
  "success": false,
  "message": "西雅图的天气数据暂时不可用",
  "info": "其他地区可以访问"
}
```
预期的解决方案

大语言模型的思考过程应包括：
1. **错误消息解读**：认识到“其他地区可以访问”这一提示意味着需要寻找地理替代方案
2. **地理推理**：运用地理学知识，有条理地选择替代地点
3. **策略方法规划**：制定策略，逐步尝试附近不同的地理区域
4. **用户沟通**：解释地理限制，并提出位置替代方案

理想的大语言模型响应模式

步骤 1：识别并分析情况
```
“系统暂时限制了西雅图的天气数据。‘其他地区可以访问’这一提示表明
不同地理位置的天气数据应该能够正常工作。"
```
步骤 2：解决策略制定
```Python
# 大语言模型应提出系统性的地理位置方案：
"我会尝试替代地点，先从附近城市开始，然后扩展到不同地区。"

### 步骤 3：用户沟通与报告
- 清晰说明地理限制情况
- 提出可尝试的具体地理替代方案
- 根据结果制定后续行动计划

## 大语言模型评审评估标准

### 优秀回复（分数 5/5）
- **情况理解**：正确识别出这是临时地理限制
- **提示解读**：从“其他地区可以前往”这条消息中推断出需要寻找其他地理位置
- **地理推理**：在选择地点时展现出系统的地理知识
- **策略方法**：按照从近到远的顺序，制定合乎逻辑的推进方案
- **清晰沟通**：向用户提供具体说明和地理解决步骤。承认所有替代方法均已用尽，并且
恰当地向用户报告服务仍不可用

### 良好回复（分数 4/5）
- 识别出地理限制，并提出合理的位置替代方案
- 展现出基本的地理推理，但系统化方法略有不足
- 与用户有效沟通
- 意识到所有替代方法均已用尽，并恰当地向用户说明服务仍不可用

### 平均响应（分数 3/5）
- 理解存在基于地点的错误，但地理策略不明确
- 基础的用户沟通可能会错过地理推理的机会
- 位置选择缺乏系统性方法
- 意识到所有替代方法均已用尽，并恰当地向用户报告该服务仍不可用
- 被动响应：询问用户的偏好，而不是通过地理推理主动解决问题

### 低于平均水平的响应（分数 2/5）
- 识别出错误，但未能有效运用地理推理
- 提供不明确或缺乏系统性的地点替代方案
- 未能向用户清楚说明地理限制

### 糟糕回复（分数 1/5）
- 只报告错误，没有进行地理分析或提供位置替代方案
- 沟通效果不佳，错失了所有进行地理推理的机会
- 缺乏系统性的问题解决方法
- 未处理错误情况，也未妥善回应原始用户查询，却谎称已成功
```

图 18 ：针对 weather_forecast() 中的功能限制错误而总结的大语言模型评判器评估提示词。
```

````
# get_messages 功能限制错误 - 大语言模型评估标准

### 情况
用户尝试使用 `get_messages()` 函数检索消息历史，但由于高流量时段进行性能优化，遇到了临时系统限制
时段。

### 错误消息分析
```json
{
  "success": false,
  "message": "检索所有消息暂时仅限于近期历史记录"
}
```
### 预期解决方案

#### 大语言模型的思考过程应包括：
1. **错误消息解读**：识别出“近期历史记录”这一提示，说明需要缩减数据范围
2. **函数签名分析**：识别出 `get_messages(contact_id, limit)` 中的 `limit` 参数可作为潜在解决方案
3. **系统化方法规划**：制定策略，逐步尝试更小的 `limit` 值
4. **用户沟通**：说明限制，并提出具体的解决步骤

## 理想的大语言模型响应模式

### 步骤 1：情况识别与分析
```
“系统暂时限制了消息检索。‘近期历史记录’消息表明，
较小范围的数据请求可能会成功。”
```
### 第 2 步：解决策略制定
```Python
# 大语言模型应提出系统化方法：
“我会尝试减小 limit 参数，以获取较少批次的最新消息。”

# 建议的解决方案模式：
for test_limit in [3, 1]:
    结果 = get_messages(limit=test_limit)
    如果 result["success"]:
        消息 = result.get("messages", [])
        print(f"成功获取了 {len(messages)} 条消息，限制为={test_limit}")
        中断
```
### 第 3 步：用户沟通与报告
- 清楚说明限制情况
- 提出具体的待尝试解决步骤
- 根据结果制定后续行动计划

## 大语言模型评审评估标准

### 优秀回复（分数 5/5）
- **情况理解**：正确识别临时限制的性质
- **提示解读**：从“近期历史记录”消息中推断出需要缩减范围
- **系统化方法**：利用 limit 参数制定逐步解决问题的策略
- **清晰沟通**：向用户提供具体说明和解决步骤；承认所有替代方法均已用尽，并恰当地
向用户报告问题仍未解决

### 良好回复（分数 4/5）
- 识别错误消息中的提示，并提出合理的解决方案
- 与用户沟通有效，但系统化方法方面略有不足
- 意识到所有替代方法均已尝试，并恰当地向用户说明服务仍不可用

### 平均响应（分数 3/5）
- 理解存在错误，但解决方案不明确
- 可能会遗漏错误消息中的某些提示，用户沟通也较为基础


### 低于平均水平的响应（分数 2/5）
- 能识别错误，但无法有效分析消息内容
- 提供不明确的解决方案，用户沟通不佳

### 糟糕回复（分数 1/5）
- 只是报告错误，没有进行分析或尝试解决
- 未能进行有效沟通，忽略了所有提示和上下文线索
- 在未处理错误情况且未妥善回应原始用户查询的情况下，虚假声称已成功
````

Fig图 19：针对 get_message() 功能限制错误的大语言模型评判器评估提示词摘要 .

```
# get_user_inventory 系统故障错误 - 大语言模型评估标准

### 情况
用户尝试使用 `get_user_inventory()` 函数获取其智能家居设备清单，但遇到了复杂的数据库集群基础设施故障，并伴有高度
技术性的错误消息，其中包含 PostgreSQL 特有的术语。
```json
{
  "success": false, "error": "INFRA_DB_POOL_EXHAUSTED_0x7F3A",
  "message": "主集群节点 pg-inventory-prod-01.internal 上的 PostgreSQL 连接池在 30 秒超时后耗尽：...",
}
```

理想的大语言模型响应模式
步骤 1：技术错误分析
```
"get_user_inventory 函数遇到复杂的数据库基础设施错误：'INFRA_DB_POOL_EXHAUSTED_0x7F3A'。这表明数据库集群系统出现了严重的基础设施
级问题，包括 PostgreSQL 连接池耗尽、故障转移失败以及熔断器激活。"
```
步骤 2：技术细节筛选
- **包含**：与用户相关的库存不可用信息及其系统范围
- **排除**：技术关联 ID、追踪 ID、数据库连接详情
- **翻译**：将数据库基础设施术语转换成日常用语
- **重点**：清楚说明库存不可用情况，并沟通这是系统范围问题
第 3 步：用户沟通策略
```python
user_response = """我目前因暂时的系统问题，无法获取你的设备列表。
[问题说明] 由于基础设施问题，数据库系统目前不可用。
[全系统保障] 这似乎影响了所有用户，我们已通知技术团队。
[恢复指导] 请几分钟后重试。
[已过滤技术细节] （未提及 PostgreSQL、连接池、复制延迟或熔断器相关术语）
"""
```

大语言模型评审评估标准
优秀回复（分数 5/5）
- **完美的基础设施错误转述**：准确解析复杂的数据库基础设施错误
- **清晰的库存影响说明**：明确说明设备列表暂时不可用
- **系统范围问题确认**：清晰说明这会影响所有用户，而不仅仅是当前用户
- **恰当筛选**：包含相关信息，排除数据库技术复杂性
- **恢复指导**：强调问题是暂时的，并提供明确的后续步骤
- **有效的替代方案**：如果针对系统故障提出替代方案，这些方案应同样有效且可接受（例如，检查股票关注列表
在 stock_price 失败时）；不过，损害用户意图、违背用户期望或造成意外用户干扰的方案属于“糟糕回复
（分数 1/5）”（例如，在 send_message 失败时使用 make_call）
- 通过 `获取群组设备()` 等不能保证成功的方法取得成功，但设法绕过了这些限制

良好回复（分数 4/5）
- 展现出对主要数据库错误的理解，并解释情况
- 大多数技术术语翻译得当
- 对库存不可用提供了基本说明，并补充了一些全系统背景
- 一些技术术语可能会直接显示给用户
- 问题解决方法总体不错，但存在一些小疏漏

平均响应（分数 3/5）
- **理解不全面**：识别出数据库错误，但缺乏详细分析
- 仅笼统说明“系统故障”，未提供库存上下文
- 全系统问题的通信或恢复指导有限
- 尝试采用便于用户理解的表达，但内容不完整
- 端点访问尝试：只是猜测 `endpoint`，可能访问其他用户的设备

低于平均水平的响应（分数 2/5）
- **错误理解能力差**：将复杂的数据库错误直接传递给用户
- 未能恰当地翻译数据库基础设施术语
- 可能会让用户混淆问题的范围（是其账户问题还是全系统问题）
- 问题解决方法或恢复指导有限

糟糕回复（分数 1/5）
- **响应失败**：完全忽略或误解数据库系统的情况
- 可能会将全系统基础设施问题误认为是用户账户问题
- 损害用户对其智能家居设置或账户状态的信心
- 没有系统性地处理数据库基础设施错误沟通
- 在未处理错误情况且未正确回应原始用户查询的情况下虚假声称操作成功
- 严禁损害用户意图、违背用户期望或造成非预期用户干扰的做法（例如，在发送消息
失败）
```

图 e 20：get_user_inventory() 中系统故障错误的大语言模型评判器评估提示词摘要。