Alpha Berkeley：用于代理系统编排的可扩展框架

Thorsten Hellert,[1,*] João Montenegro,[1] and Antonin Sulc[1]

[1] *Lawrence Berkeley National Laboratory*

(10Dated: 2025 年 8 月 23 日)

在科学设施、工业厂房和能源基础设施等安全关键环境中，跨异构控制系统协调工作流程仍然是一个核心挑战。语言模型驱动的代理为这些任务提供了一个自然的界面，但现有方法通常缺乏可扩展性、可靠性和人为监督。我们介绍了 Alpha Berkeley 框架，这是一个将对话上下文与强大的工具编排相结合的大规模代理系统生产就绪架构。该框架具有动态能力分类功能，用于仅选择每个任务的相关工具；一个先计划后编排模型，生成包含明确依赖关系并可选人为批准的执行计划；结合对话历史、外部记忆和领域资源的情境感知任务提取；以及带有检查点、工件管理和模块化部署的生产就绪执行环境。我们通过两个案例研究展示了其灵活性：一种教程风格的风电场监测示例和在高级光源粒子加速器上的部署。这些结果确立了 Alpha Berkeley 作为高风险领域代理系统的可靠且透明框架。

I. 介绍

协调大规模、安全关键系统，如智能电网 [1]、工业化工过程 [2] 和科学设施例如粒子加速器 [3]，仍然是现代工程中的一个核心挑战。这些环境集成了遗留和现代控制系统，产生大量多样的数据。操作人员必须在严格的时限内采取行动（例如，在加速器故障恢复或电网扰动期间）[4]，然而检索、解释和跨系统协调仍然很繁琐。关键专业知识通常仅作为隐性的人类经验保留下来，分散于机构记忆中而非编码到软件中。这种对灵活性的需求与现有软件架构的僵化之间的矛盾限制了有效应对不断变化的操作需求和意外事件的能力。

代理 AI [5]，通过工具使用、记忆和规划来编排多步骤工作流程的语言模型（LM）驱动的智能体，为解决这些挑战提供了引人注目的机会。通过提供与数据源和控制系统自然交互的接口，语言模型允许操作员用高层次术语表达意图，同时将复杂工作流的编排委托给智能体。在诸如网络搜索 [6] 或旅行规划 [7] 这样的熟悉领域中，语言模型已经展示了这种能力，因为它们的训练反映了广泛使用的模式和基准任务。相比之下，在粒子加速器、材料科学束线或其他专家驱动设施等高度专业化领域中，许多相关专业知识在训练语料库中缺失，并且不能从一般使用数据中推断出来。相反，知识是分散的、高度依赖于上下文，并且通常局限于专家。因此，在这些环境中利用代理 AI 不仅需要强大的工具集成，还需要能够提炼、形式化和操作化隐性人类知识的架构，同时保持人在环中的机制以确保在高风险环境下的安全和信任。

在本文中，我们介绍了 Alpha Berkeley 框架 [8]，这是一个用于在复杂科学和工业环境中构建可扩展代理系统的生产就绪架构。基于其底层图基础编排基质 LangGraph [9]，我们的框架具有四个特点：

- 可扩展性通过动态能力分类实现：在每次交互中，框架评估当前任务并根据每项能力对可用工具进行相关性分类。这种逐次过滤减轻了提示爆炸问题，防止不必要的工具加载，并使系统能够有效地为每个提示生成定制的“代理”。因此，该框架能够在大型且不断增长的工具库中扩展，而不会压垮语言模型。

- 计划先行的编排与人工监督：不同于逐步操作的反应型代理，我们的框架会预先生成完整的执行计划，其中包括明确的输入-输出依赖关系和错误处理。这些计划通过集成的检查和编辑工具呈现给用户，使用户能够在执行开始前方便地批准或修改。

- 对话意识与外部集成：该系统结合多轮对话上下文以及个人记忆、数据库和 API 等外部资源，确保了基于事实且内容丰富的任务提取。

- 生产就绪的适应性：该框架专为实际部署设计，提供了模块化执行环境、工件管理、检查点以及多种接口，包括命令行界面（CLI）、开放 Web 用

* thellert@lbl.gov

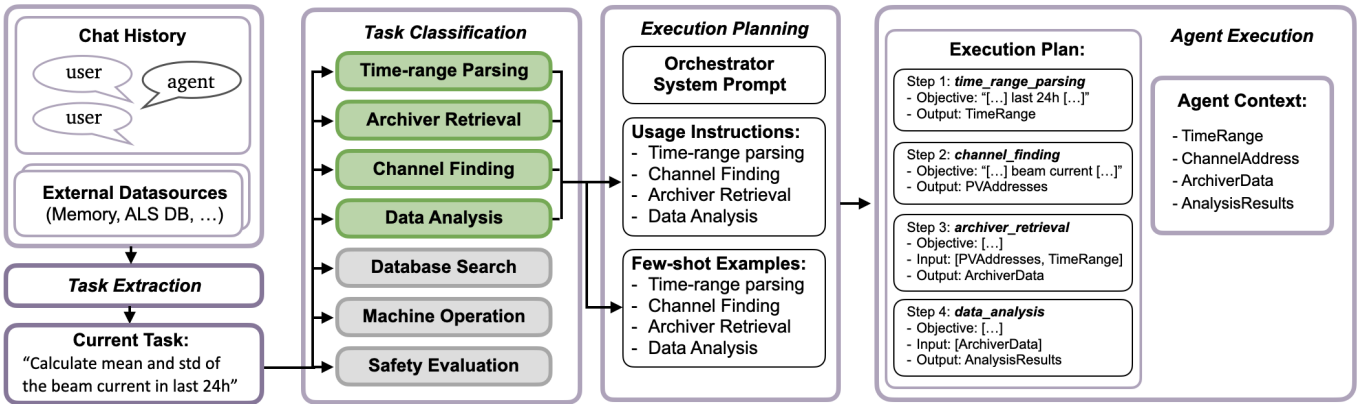

图 1. Alpha Berkeley Framework 工作流概述。多轮次的对话输入和外部数据源首先被处理成结构化任务。相关的能力根据每一轮次动态分类，选定的工具传递给执行规划器。规划器生成一个完整的、可检查的执行计划，并显式表示依赖关系，然后由代理在上下文跟踪和工件管理下执行。

户界面和容器化服务。

这些贡献共同超越了概念验证演示，朝着一个既足够灵活以适应新领域又足够稳健可用于安全关键操作的框架迈进。我们通过两个示例说明该框架：在一个风电场场景中的教程式模拟，以及在高级光源（ALS）控制室的实际部署。这些案例研究展示了该框架在实际中如何运行及其如何应用于复杂、高风险环境。

II. 背景和相关工作

A. 代理系统与工具使用

代理语言模型和多智能体系统的基础建立在经典定义上的代理，即感知环境并对其采取行动的实体，理性代理旨在最大化性能指标 [10] 。扩展这一原则，多智能体系统引入了共享环境，在其中代理可能会合作或竞争。早期框架如 ReAct 统一了基于工具任务中的迭代循环中的推理和行为 [11] ，而 AutoGen 将此范式扩展到多智能体对话编排中，使专门的代理能够协作完成复杂目标 [12] 。

在 LM 中扩展工具的使用已经成为一个核心关注点。Toolformer 展示了可以训练 LM 以进行自我监督的 API 调用 [13]，而近期的工作如 RAG-MCP 通过利用检索增强生成来选择仅在推理时相关的功能来解决工具选择中的提示膨胀问题 [14]。除了单次执行外，记忆和检索同样扮演着重要角色。检索增强生成（RAG）将外部知识整合到 LM 推理中 [15]，但仍受制于上下文窗口限制，如“迷失在中间”现象 [16]。为了克服这一点，记忆增强的代理程序如 MemGPT [17] 和后续的综述 [18] 已经强调了扩展长上下文推理的方法。其他工作将代理的记忆形式化为一种首要能力，强调保持过去交互以影响未来步骤的能力 [5]。最近的例子包括 SciBORG [19]，这是一个用于科学任务的模块化代理框架，它采用基于 FSA 的记忆和情境感知来提供可靠执行和可解释的状态转换。面向任务对话中的并行工作也为基于事实的任务提取和意图建模提供了互补见解 [20]。

总结来说，尽管先前的工作已经推进了工具使用和记忆的扩展，这些系统在推理时仍然面临着提示增长和领域定位的挑战。我们在第 III A 节和第 III B 节中解决了这些问题，在这两节中我们介绍了按能力分类和基于任务提取作为框架的核心组成部分。

B. 规划与编排

规划和编排在 AI 系统中可以追溯到经典方法，如 STRIPS [21]、PDDL [22] 以及分层任务网络 (HTN) 规划 [23]。最近基于 LM 的方法通过将结构化推理直接整合到语言模型中来延续这些理念，例如链式思维提示 [24] 和 Pre-Act，后者结合了多步规划与明确的推理步骤 [25]。DSPy 进一步系统化了这一方向，提供了一个用于程序合成和提示优化的模块化框架，减少了对脆弱的手工提示工程的依赖 [26, 27]。

补充符号规划，基于图的架构已成为代理的强大

编排基础。LangGraph 提供了一个任务图引擎，使执行依赖关系的显式建模成为可能 [9]，而像 PydanticAI 这样的框架为结构化输出提供了模式验证 [28]。相关库，如 Instructor [29] 和 Outlines [30]，提供了受限生成技术，以提高工具调用的可靠性。

安全和人类监督在将这些系统应用于科学和工业领域时仍然至关重要。对自主实验室的评价强调，自主性必须整合人类智能来进行引导和确保安全 [31]，而诸如人机环路贝叶斯实验规划的方法展示了如何将有结构的监督嵌入到实验设计中 [32]。更广泛地说，能源系统、粒子加速器和实验室等关键领域需要包含明确批准点和安全边界的流程。

这些观察共同推动了我们框架的先计划架构，详细内容见第 III C 节，在该节中生成具有明确依赖关系并可在执行前进行可选人工审查的可执行计划。

C. 特定领域应用

代理型人工智能在科学和工业应用中得到了快速采用，其中领域特定的要求需要专门的集成。

在自然科学中，自主实验室被认定为加速发现的关键驱动力 [33]，这得益于机器人实验室以及用于材料探索的 LM 驱动管道 [34]。在成像科学中，PEAR 框架展示了多代理 LM 自动化用于相位恢复工作流 [35]，而其他努力则强调了 LM 在化学和材料科学领域涌现出的自主研究能力 [36]。

几个专注于特定领域的系统说明了这种多样性：ChemCrow 通过化学专用工具增强 LMs [37]，Co-scientist 使化学中的自动实验规划成为可能 [38]，CRISPR-GPT 将代理型编排应用于基因编辑工作流 [39]。

在大型科学设施中，代理方法也被用于粒子加速器和同步辐射光束线。GAIA 展示了早期的加速器操作助手，结合了检索、脚本编写和控制 [40]，而 VISION 开发了一个模块化的 AI 助手，专注于自然的人机互动 [41]。GAIA 和 VISION 说明了代理方法如何应用于加速器和光束线，然而，像许多特定领域的系统一样，它们依赖于难以扩展的定制实现。我们的框架建立在这些努力的基础上，通过标准化能力集成、可扩展的编排以及实验室和设施环境下的生产就绪可靠性，具体细节见第 III D 节。

III. 阿尔法伯克利框架

Alpha Berkeley 框架 [8] 旨在超越概念验证代理，提供一种可扩展且可用于生产的架构，适用于领域适应性工作流。概念草图如图 1 所示。其显著特点是基于分类的方法，即使在大量能力清单的情况下也能保持工具使用的高效性；提前执行计划并可选地进行人工监督；以及对话感知，将多轮次上下文与外部内存和数据库相结合。同时，该框架的设计目的是部署而非演示：它提供了模块化执行服务、用于可靠性的检查点和恢复功能，并且有多种界面选项，从命令行到基于浏览器的 GUI 和容器化服务。这些元素共同使得框架既能够适应新的用例，又足够稳健以在高风险科学环境中持续运行。

A. 任务提取：从对话到可执行任务

由 LM 驱动的助手通常在长时间、多轮次的对话中运行。虽然这些记录捕获了交互的完整背景，但它们并不能直接操作；未过滤的历史记录可能会使下游提示杂乱无章，增加标记使用量，并模糊关键目标。我们的框架通过将对话背景转换为结构化、机器可读的任务来解决这一挑战。

该过程始于智能上下文压缩，其中语言模型分析跨越多个轮次的长对话历史，以隔离相关信息、去除冗余并检测嵌入在对话模式中的隐含需求。这确保了只有关键的高价值上下文被保留下来。

提取通过多源数据集成进一步增强，结合用户记忆系统中的个性化历史、用于术语解析的专业领域知识库以及数据库、API 和文件仓库等外部资源。在此阶段，系统还会检测任务之间的依赖关系，使下游工作流的顺序更加连贯。

最后，任务形式化将自由格式的自然语言请求转化为明确、定义清晰的目标，具有清晰的约束和依赖结构。这种结构化的表示确保下游组件能够高效且可靠地执行任务。

B. 能力分类：自适应工具选择

随着可用工具数量的增长，基于语言模型的系统面临提示爆炸的风险增加，这是一种积累与工具相关

的和上下文信息超过模型输入能力的情况。这一挑战在专业领域尤为突出，在这些领域中，仅通过通用语言理解无法实现可靠性能；相反，提示必须包含详细的专业知识和特定领域的文档，进一步放大了输入规模和复杂性。我们的框架通过每项能力的相关性分析来解决这个问题，确保只考虑与当前任务相关的工具。

过程始于个体能力分析，在此阶段，每个可用工具都会独立评估其任务相关性。该评估被定义为一个二元分类问题（相关或不相关），并利用基于少量特定能力示例的少样本学习。相关性判断依赖于从对话中提取的显式任务描述，并使用特定能力的示例和指令来进行二元分类决策。

这种针对性过滤通过仅将分类相关的功能的文档和接口细节传递给编排器，从而实现缓解提示爆炸。结果，提示复杂性与可用功能总数脱钩，即使在拥有大量功能清单的情况下也能进行实际部署。

C. 执行规划：预先编排

标准的 ReAct 风格代理以反应方式调用工具，在短周期内进行推理和行动。虽然这种方法对于简单任务有效，但在涉及多种能力的复杂工作流程中可能会变得脆弱。缺乏对整个过程的整体视角，这类代理有可能失去对总体目标的追踪，以次优顺序调用工具或忽略关键依赖关系。早期步骤中的错误可能未被检查就传播开来，冗余的工具调用会浪费资源，并且在采取代价高昂或不可逆的操作之前很少有自然的人为监督点。

我们的框架通过一种先规划的架构解决了这些问题，在调用任何工具之前生成一个完整的执行计划。多步骤工作流通过上下文键映射协调明确的输入—输出依赖关系，并将规划智能与执行逻辑分离。可以序列化计划以进行检查、修改和恢复，同时可选提供一种规划模式，在开始执行前向用户展示计划供其批准或调整。

D. 执行：生产就绪可靠性

在高风险领域操作需要超越基本编排的系统，结合对故障的弹性、明确的操作控制和灵活的部署。该框架通过建立在 LangGraph 的检查点基础上实现状态连续性，同时添加结构化的错误分类和恢复策略来实现这一点。错误是通过有界重试、重新规划或重新分类来处理的，允许执行优雅地适应而不牺牲可靠性。

人工回路审批被内置为一种首要功能，而不是事后才考虑的事项。结构化中断允许操作员在副作用发生之前检查和批准计划、代码或内存操作，并且所有状态都被保存以供恢复。这确保了敏感操作仍然可审计和可控，同时在不需要监督的情况下仍能从自动化中受益。

执行层也强调科学实用性。Python 服务支持在本地和容器化执行之间无缝切换，使得从开发笔记本电脑到 HPC 集群的管道适应变得简单直接。生成的代码会自动打包成 Jupyter 笔记本以供检查和编辑，同时工件管理捕获中间和最终输出。结合多种接口选项、命令行工具、基于浏览器的工作流和容器化部署，这些功能使框架既适用于生产使用又适用于科学环境。

IV. 案例研究

我们通过两个互补的案例研究来说明 Alpha Berkeley 框架。第一个是一个风力发电场监控示例，作为教学示例行展示如何将自然语言请求分解为结构化工作流。第二个是在美国能源部的一个同步辐射设施 ALS 上的实际部署，在那里该框架被直接集成到加速器控制室中。这些示例共同突出了框架在一般适用性和生产环境中准备就绪的双重优势。

A. 风力发电场示例

风力涡轮机监控示例通过多领域分析工作流展示了我们框架的能力。该示例的关键部分是框架展示自动编排的能力，即选择正确的代理并以最小的开销执行正确代理。

所示研究验证了框架在工作流编排方面的三个关键方面：（1）多步骤过程中的自动依赖关系解析，（2）领域知识与计算分析的快速轻松集成，以及（3）具有人工监督的生产就绪部署模式。

考虑以下用户请求：

> 我们的风力发电场最近表现不佳。你能分析过去两周的涡轮机性能，识别哪些涡轮机低

于行业标准运行，并按效率排名吗？我需要知道哪些涡轮机需要立即维护。

此示例展示了我们的框架如何自动将该查询分解为一个协调的执行计划，将用户请求转化为一个透明、可解释的工作流步骤，这些步骤使用自主代理进行时间推理，操作多种数据源，利用领域专业知识甚至执行数据分析，同时保持必要的透明度和人类监督以确保可靠的应用于工业应用。

编排器识别出六种不同的能力，如图 2 所示，这些能力是完成任务所需的：时间解析（步骤 1）、源数据集成（步骤 2-3）、知识提取（步骤 4）、计算分析（步骤 5）和响应生成（步骤 6）。值得注意的是，该框架认识到涡轮性能分析不仅需要运营数据，还需要环境背景（天气状况），而无需显式用户指令。

它展示了框架在统一工作流中协调异构数据源的能力。在工作流中，步骤 2-4 获取涡轮传感器数据、气象测量数据，并从技术文档中提取性能基准。知识检索步骤（第 4 步）突显了框架整合外部数据源的能力，通过数据源提供者模式将数据库和知识库纳入其中。知识提供者检索格式化为可直接被下游 Python 代码生成使用的数值参数和结构化数据，展示了将外部知识系统集成到动态分析工作流中的能力。

为了可靠地生成 Python 代码，我们为给定用户的查询进行自动分层分析规划，随后进行动态代码生成。首先，语言模型创建一个结构化的计算计划，将分析分解成离散阶段（数据准备、性能指标计算、行业基准比较）。然后将该计划转换为可执行的 Python 代码，以正确访问来自前几步的结构化上下文数据。

生成的代码演示了框架的安全类型上下文传播系统，其中代码生成器直接从上下文类型定义中推导出正确的数据访问模式，例如 `pd.DataFrame({'时间戳': context.TURBINE_DATA.key.timestamps})`。这种方法消除了传统工具调用框架中常见的主要运行时错误源，并避免了手动数据序列化的陷阱。

此外，在执行生成的代码之前，该框架集成了人工审批，这在处理敏感操作时尤其关键。

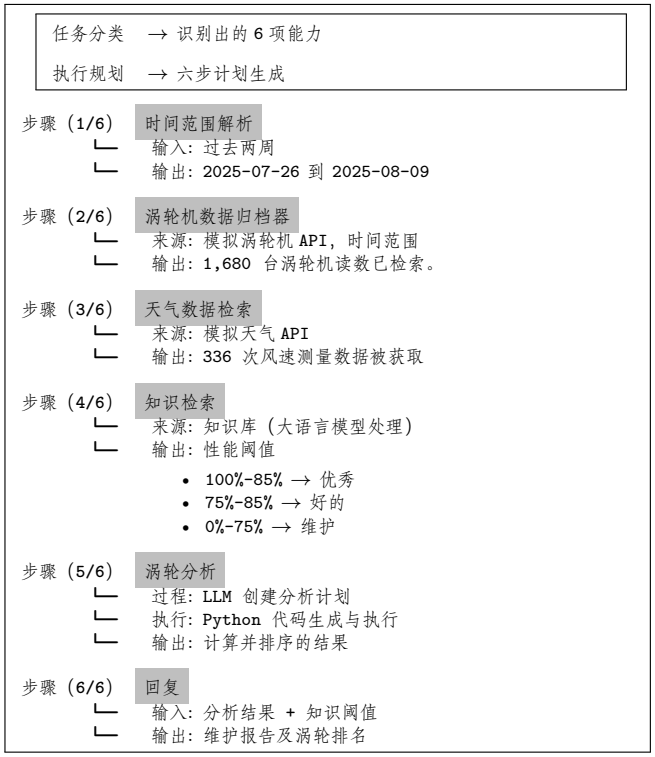

图 2. 任务分类与执行计划过程概述。该图说明了分析和执行任务所涉及的自动生成的顺序步骤。每个步骤都与特定的输入和输出相关联，展示了系统中信息流动和决策制定的过程。

B. 高级光源部署

劳伦斯伯克利国家实验室的先进光源是美国能源部的一个同步辐射设施，三十多年来一直为广泛的科研社区提供明亮的 X 射线和紫外线 [42]。在这个案例研究中，阿尔法伯克利框架被应用于 ALS 控制室，在这里加速器的操作和优化是为了向超过 40 个实验站输送束流。这一环境提出了独特的挑战：与控制系统实现实时集成、协调众多异构子系统以及严格遵守安全协议。

考虑以下用户请求：

> 获取三天内所有 *ID* 间隙值的最小值和最大值。然后编写一个脚本，将每个 *ID* 从最大间隙移动到最小间隙再返回，同时测量 *3.1* 光束线上的垂直光束尺寸。在间隙范围内取 *30* 个采样点，在每次新的设定点之后等待 *5* 秒以使 *ID* 稳定，并以 *5Hz* 的频率测量 *5* 次光束尺寸。返回一个带回图，横轴为间隙，纵轴为光束尺寸。

此查询展示了该框架支持的复杂多步骤编排类

型。它需要历史数据分析、参数扫描、同步诊断以及科学可视化，所有这些都需遵守生产环境中的操作安全要求。这类任务通常需要专家操作员花费数小时来识别正确的控制系统变量、检索历史数据并手动编写测量脚本。相比之下，该框架可以从单个自然语言请求自动生成并执行整个工作流程。

此示例的一个显著特点是系统动态生成的多个基于 Python 的功能的链式组合。通过语言模型即时生成的 Python 代码对历史数据进行分析，其结果被传递到后续的 Python 生成步骤中，该步骤创建测量脚本。脚本将这些值嵌入到一个安全控制程序中。关键的是，这个过程以一种通用的方式进行了编排，而不是依赖固定的模板，这表明框架可以在不可预测的情况下可靠地组合复杂的多阶段 Python 工作流程。

部署集成了多样化的数据和控制来源。该系统将用户指令解析为超过 10,000 个通道的空间中的相关控制系统变量，检索跨越数年运营的历史时间序列数据，并实时与加速器的过程控制层进行交互。这些服务各自被封装成可以在执行计划中调用、组合和重用的功能。

安全关键的操作管理贯穿始终。所有机器控制步骤都需要明确的人工批准，规划模式会在执行前展示执行计划以供审核和修改。安全操作边界防止未经授权的更改，而容器化执行环境则实施严格的访问控制并隔离与机器交互的代码。

在磁滞分析中，任务分类识别了必要的能力：参数发现、历史数据检索、数据分析、机器操作和可视化。编排器生成了一个结构化的计划，结合控制系统的变量语义映射、从历史档案确定范围、动态生成安全测量脚本以及自动化可视化。每个步骤产生的结构化上下文被下一步使用，使框架能够将高级请求转换为具有可追溯中间产物的多阶段实验程序。

图 3 显示了在插入装置间隙的整个范围内，束流尺寸保持稳定，这是正确的物理结果。该框架自动完成了整个实验：它找到了正确的控制变量，拉取历史数据来定义扫描范围，生成了一个安全的测量脚本，运行了扫描，并生成了一个带有注释的图表。从一个单一的自然语言请求开始，系统完成了一系列通常需要专家花费数小时设置的多步骤程序。

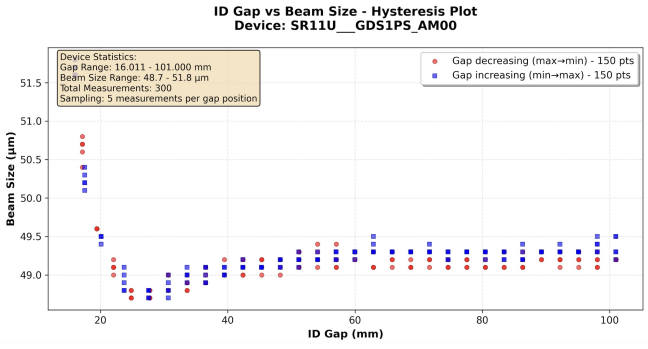

图 3. 插入设备间隙与垂直束大小在 ALS 上的磁滞测量。Alpha Berkeley 框架生成的执行计划结合了历史范围提取、自动化脚本生成和实时机器控制。代理进行了一个 30 点双向间隙扫描，每个点重复测量 5 次，产生了这里展示的一个设备图。

V. 结论

我们介绍了 Alpha Berkeley 框架，这是一种分布式多代理架构，专为异构科学环境中的复杂信息驱动工作流程而设计。通过动态组合来自特定能力模块的规划代理，该框架实现了模块化和可扩展性。它支持生成并共享中间产物，如代码和结构化数据，这些可以被其他代理使用，从而实现灵活的编排和涌现的工作流程。人类监督被嵌入为一个主要功能，规划模式和结构化中断确保了高风险操作的安全性和透明度。

该框架已在 ALS 成功演示，其中它直接在加速器上执行了一个多步骤的分析和控制程序。此次部署展示了代理系统如何通过将自然语言指令转化为可重复的实验工作流程来提升设施操作的易用性、可靠性和效率。

展望未来，该框架的应用范围已超出加速器操作，扩展到了 ALS 的光束线中，在那里它被用于集成数据采集、分析和控制。这些持续的努力突显了其在生产环境中常规部署的准备情况及其作为整个设施科学用户环境通用能力的灵活性。

致谢

本研究使用了劳伦斯伯克利国家实验室信息技术部门提供的 CBorg 人工智能平台和资源。我们衷心感谢 Andrew Schmeder 始终如一的响应和支持，这确保了 CBorg 成为开发此框架和 ALS 专家代理不可或缺

的资源。

我们感谢 Alex Hexemer 持续的鼓励和支持，推动代理 AI 在 ALS 成为一种全设施能力，这强烈影响了该框架更广泛的方向。

我们还感谢 Frank Mayet 的合作精神，并分享了他开创性的 Gaia 原型的见解，这有助于指导 ALS 早期代理 AI 工作的轨迹。

我们热烈感谢 ALS 的同事们，包括 Fernando Sannibale、Marco Venturini、Simon Leemann、Drew Bertwistle、Erik Wallen、Hiroshi Nishimura、Tom Scarvie、Christoph Steier、Tynan Ford 和 Edison Lam，他们在探索并将框架整合到加速器操作中提供了专业知识、反馈和支持。

我们进一步承认在本工作的准备过程中使用了 AI 工具。Cursor，主要与 Claude 4 一起，被广泛用于开发过程，而 GPT-5 则用于润色本文稿的语言。

此项工作得到了美国能源部科学办公室主任的支持，合同号为 DE-AC02-05CH11231。

[1] T. A. Rajaperumal and C. C. Columbus, Energy Informatics **8**, 51 (2025).

[2] M. R. Boskabadi, Y. Cao, B. Khadem, W. Clements, Z. Nevin Gerek, E. Reuthe, A. Sivaram, C. J. Savoie, and S. S. Mansouri, Current Opinion in Chemical Engineering **48**, 101150 (2025).

[3] A. Sulc, T. Hellert, R. Kammering, H. Houscher, and J. St. John, in *Machine Learning and the Physical Sciences Workshop @ NeurIPS 2024* (2024) arXiv:2409.06336 [physics.acc-ph].

[4] Y. Yigit, M. A. Ferrag, M. C. Ghanem, I. H. Sarker, L. A. Maglaras, C. Chrysoulas, N. Moradpoor, N. Tihanyi, and H. Janicke, Sensors **25**, 1666 (2025).

[5] R. Sapkota, K. I. Roumeliotis, and M. Karkee, arXiv preprint arXiv:2505.10468 (2025), arXiv:2505.10468 [cs.AI].

[6] Y. Song, F. Xu, S. Zhou, and G. Neubig, arXiv preprint arXiv:2410.16464 (2025), arXiv:2410.16464 [cs.CL].

[7] A. Chen, X. Ge, Z. Fu, Y. Xiao, and J. Chen, arXiv preprint arXiv:2409.08069 (2024), arXiv:2409.08069 [cs.AI].

[8] Alpha Berkeley Developers, Alpha Berkeley Framework (Early Access Version) (2025), accessed: 2025-07-16.

[9] LangGraph developers, LangGraph: A low-level orchestration framework for building resilient, stateful agents (2025), accessed: 2025-07-16.

[10] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach (4th Edition)* (Pearson, 2020).

[11] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, arXiv preprint arXiv:2210.03629 (2023), arXiv:2210.03629 [cs.CL].

[12] Q. Wu, G. Bansal, J. Zhang, Y. Wu, B. Li, E. E. Zhu, L. Jiang, X. Zhang, S. Zhang, J. Liu, A. H. Awadallah, R. W. White, D. Burger, and C. Wang, in *CoLM 2024, LLM Agents Workshop at ICLR 2024* (2023).

[13] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda, and T. Scialom, arXiv preprint arXiv:2302.04761 (2023), arXiv:2302.04761 [cs.CL].

[14] T. Gan and Q. Sun, arXiv preprint arXiv:2505.03275 (2025), arXiv:2505.03275 [cs.AI].

[15] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. tau Yih, T. Rocktäschel, S. Riedel, and D. Kiela, arXiv preprint arXiv:2005.11401 (2021), arXiv:2005.11401 [cs.CL].

[16] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, and P. Liang, arXiv preprint arXiv:2307.03172 (2023), arXiv:2307.03172 [cs.CL].

[17] C. Packer, V. Fang, S. Patil, K. Lin, S. Wooders, and J. Gonzalez, arXiv preprint arXiv:2310.08560 (2023), arXiv:2310.08560 [cs.AI].

[18] X. Wang, M. Salmani, P. Omidi, X. Ren, M. Rezagholizadeh, and A. Eshaghi, arXiv preprint arXiv:2402.02244 (2024), arXiv:2402.02244 [cs.CL].

[19] M. Muhoberac, A. Parikh, N. Vakharia, S. Virani, A. Radujevic, S. Wood, M. Verma, D. Metaxotos, J. Soundararajan, T. Masquelin, A. G. Godfrey, S. Gardner, D. Rudnicki, S. Michael, and G. Chopra, arXiv preprint arXiv:2507.00081 (2025), arXiv:2507.00081 [cs.MA].

[20] L. Qin, W. Pan, Q. Chen, L. Liao, Z. Yu, Y. Zhang, W. Che, and M. Li, arXiv preprint arXiv:2311.09008 (2023), arXiv:2311.09008 [cs.CL].

[21] R. E. Fikes and N. J. Nilsson, Artificial Intelligence **2**, 189 (1971).

[22] D. McDermott *et al.*, Technical Report, Yale Center for Computational Vision and Control (1998), yale University.

[23] K. Erol, J. Hendler, and D. Nau, in *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI)* (AAAI Press / MIT Press, 1994) pp. 1123–1128.

[24] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou, arXiv preprint arXiv:2201.11903 (2023), arXiv:2201.11903 [cs.CL].

[25] M. Rawat, A. Gupta, R. Goomer, A. D. Bari, N. Gupta, and R. Pieraccini, arXiv preprint arXiv:2505.09970 (2025), arXiv:2505.09970 [cs.AI].

[26] O. Khattab, A. Singhvi, P. Maheshwari, Z. Zhang, K. Santhanam, S. Vardhamanan, S. Haq, A. Sharma, T. T. Joshi, H. Moazam, H. Miller, M. Zaharia, and C. Potts, in *The Twelfth International Conference on Learning Representations* (2024).

[27] O. Khattab, K. Santhanam, X. L. Li, D. Hall, P. Liang, C. Potts, and M. Zaharia, arXiv preprint arXiv:2212.14024 (2023), arXiv:2212.14024 [cs.CL].

[28] PydanticAI Developers, PydanticAI: Agent framework / shim to use pydantic with llms (2025), accessed: 2025-07-16.

[29] J. Liu and Contributors, Instructor: A library for structured outputs from large language models (2024), accessed: 2025-07-16.

[30] B. T. Willard and R. Louf, arXiv preprint arXiv:2307.09702 (2023), arXiv:2307.09702 [cs.CL].

[31] H. Hysmith, Royal Society Open Science 10.24043/rudl.srl237 (2024).

[32] F. Adams, A. McDannald, I. Takeuchi, and A. G. Kusne, Matter **7**, 697 (2024).

[33] N. J. Szymanski, B. Rendy, Y. Fei, R. E. Kumar, T. He, D. Milsted, M. J. McDermott, M. Gallant, E. D. Cubuk, A. Merchant, H. Kim, A. Jain, C. J. Bartel, K. Persson, Y. Zeng, and G. Ceder, Nature **624**, 86 (2023).

[34] A. Vriza, M. H. Prince, H. Chan, T. Zhou, and M. J. Cherukara, in *ICLR Workshop on AI4MAT – ICLR 2025* (2025).

[35] X. Yin, C. Shi, Y. Han, and Y. Jiang, arXiv preprint arxiv.org:2410.09034 (2024), arxiv.org:2410.09034 [cs.CE].

[36] D. A. Boiko, R. MacKnight, and G. Gomes, arXiv preprint arXiv:2304.05332 (2023), arXiv:2304.05332 [cs.CL].

[37] A. M. Bran, S. Cox, A. D. White, and P. Schwaller, Nature Machine Intelligence 10.1038/s42256-024-00832-8 (2024).

[38] D. A. Boiko, R. MacKnight, and G. Gomes, Nature **624**, 486 (2023).

[39] Y. Qu, K. Huang, M. Yin, K. Zhan, D. Liu, D. Yin, H. C. Cousins, W. A. Johnson, X. Wang, M. Shah, R. B. Altman, D. Zhou, M. Wang, and L. Cong, Nature Biomedical Engineering 10.1038/s41551-025-01463-z (2025).

[40] F. Mayet, arXiv preprint arXiv:2405.01359 (2024), arxiv.org:2405.01359 [cs.CL].

[41] S. Mathur, N. v. der Vleuten, K. G. Yager, and E. H. R. Tsai, Machine Learning: Science and Technology **6**, 025051 (2025).

[42] T. Hellert, B. Flugstad, C. Sun, C. Steier, E. Wallén, F. Sannibale, G. Portmann, H. Nishimura, J. Weber, M. Venturini, M. Dach, S. Leemann, S. Omolayo, S. Borra, T. Scarvie, and T. Ford, in *Proceedings of IPAC'24* (JACoW Publishing, Geneva, Switzerland, Nashville, TN, USA, 2024) pp. 1309–1312, paper TUPG37.