ARI: 自主研究基础设施

树神宇德

v0.9.0, 2026-06-12

摘要

本报告描述了一个自主研究基础设施 **ARI**, 它将基于树搜索的探索循环与论文生成流水线相结合。探索循环是对研究步节点的最佳优先树搜索 (BFTS); 节点扩展由 Reason-and-Act (ReAct [1]) 风格的智能体执行, 该智能体向 14 个领域技能的注册表发出工具调用。论文流水线将存活的谱系汇编为草稿, 并在层次化评分细则下迭代改稿。所有产物都驻留在单一的检查点目录中, 该目录是可复现性、谱系与成本核算的单位。本报告形式化算法与设计依据, 报告初步观察, 并讨论制约设计的确定性与新颖性的权衡。

1 引言

1.1 动机

机器学习及相邻计算领域的研究流程具有共同骨架: 构思研究构想、设计并执行实验、分析结果、撰写论文, 然后迭代。其中每一步对大语言模型 (LLM) 而言都日益可行, 如今已发表的系统也已将该循环的大段环节自动化。AI Scientist [2, 3] 闭合了从构想到论文的完整循环, 并以自动审稿器为结果打分;Agent Laboratory [4] 在 LLM 评判的闸门下执行人类提供的构想; AIDE [5] 将工程内循环建模为对候选解的树搜索, MLE-bench [6] 发现在相同模型条件下该范式胜过通用脚手架;VirSci [7] 表明多智能体的协商比单一智能体产生更新颖的研究构想, 但止步于摘要;Story2Proposal [8] 在持久的结构契约下, 将已完成的实验证据转化为手稿; 而 PaperBench [9] 则事后衡量智能体复现已发表论文的忠实程度。

这些系统都未曾将所生成论文所陈述的内容与其实验所产出的结果之间的关系作为强制约束。验证若存在, 也仅是建议性的: AI Scientist 的审稿器为成稿打分, 而幻觉控制仍停留在提示词层面 — 其作者记录了误报的硬件, 以及被当作改进呈现的实则恶化的指标, 并将 “对所报告结果进行更深入的自动验证” 列为关键的未来工作 [2];Agent Laboratory 的模拟审稿器在十分制上把人类审稿评分高估了 2.3 分 [4]; 而 PaperBench 的核心发现是一道执行鸿沟 — 智能体因撰写代码获得了可观的分数, 却几乎无法因执行代码并匹配论文结果而得分 [9]。

ARI 的存在正是为了弥合这道鸿沟。用户提供研究问题与算力预算; 系统返回可发表的论文、支撑每项主张的实验代码, 以及完整的审计轨迹: 搜索树的每一个节点、每一次工具调用、每一次模型调用, 以及每一美元的 API 花费, 均记录于单一的检查点目录之中 — 它是可复现性的单位。

1.2 以验证过的主张作为发布条件

ARI 的标志性主线贯穿整条流水线。在构想阶段, 运行会铸造一份指标契约: 即最终论文必须支撑的可证伪主张, 每一项都指明可作为证据的确切测量名称。该契约一经铸造便冻结, 因为 LLM 的命名无法在重新生成后存续 (section 4.3)。在探索期间, 节点以这些名称发出测量值, 这些名称引导扩展朝向尚未覆盖的主张, 而谱系串联则让由早期测量计算而来的证据 — 一次拟合、其留出验证、一次基于模型的选择 — 沿单一谱系执行, 而不必退化为反复的探测 (section 3.8)。在撰写阶段, 每一处数值提及都绑定到它所依据的节点与测量, 而一道确定性的 claim-evidence 闸门 — 回路中没有任何语言模型 — 会从记录的结果重新推导每一个所报告的数字。客观的虚假陈述在最终闸门处阻止发布; 而互补的语义审查所给出的主观判断仅为精炼循环提供建议。

1.3 贡献

本报告对以下四项贡献加以形式化:

1. 由**确定性 claim-evidence 闸门**强制执行的 **idea 持有指标契约** (section 4.3): 验证作为流水线的一项结构性属性, 而非审稿器对其输出的意见。

2. 在研究步节点上的**最佳优先树搜索** (BFTS) (section 3.1), 其选择与扩展由 LLM 在结构化候选描述上驱动, 并配有确定性的回退排序 (section 3.3)。

3. 由层次化评分细则 R 驱动的**论文生成流水线**; 其叶判官对各项主张进行评分, 系统评分为 $\text{score}(P) = \sum_i w_i \, \text{judge}_i(P)$ (section 4.4)。PaperBench 复现器作为工具调用集成 (section 4.6)。

4. **检查点作用域**纪律: 每一项外部状态 (记忆、成本账本、谱系指针) 都写入 `$ARI_CHECKPOINT_DIR` 之下, 绝不进入用户主目录。这是确定性设计原则的运维落地, 而该原则正是 section 2.5 完整列出的五项之一 (我们在该处及本报告余下部分将其称为 P2)。

1.4 本报告的范围

本报告描述 `ari-core` 与十四个 `ari-skill-*` 包的 v0.9.0 版本, 其主题是端到端的主张验证:section 4.3 中一次铸造即冻结的契约与闸门加固; 谱系串联 (section 3.8); 一条可选启用的构思路径, 它在实时文献快照上原封不动地驱动已发表的 VirSci 协商引擎 (section 6.2); 平台感知的构思, 将探测到的平台能力事实折入构想生成, 从而使计划不能承诺执行平台并不提供的测量; 以及一篇已发布的、其自身经过闸门验证的样例论文, 其所声明的全部十二项可证伪主张均携带经机器验证的证据 (section 5)。这些新增内容建立在 v0.8.x 的基础之上 — 协议忠实的三阶段 PaperBench 再现契约 (智能体 rollout、再现、评分)、宿主真实的环境护栏, 以及可配置的搜索-评估层;section 5 中报告的针对某个非可逆压缩对象的操作性再现阶段, 运行于当时的 v0.8.0 流水线。正文以模块层级描述算法与数据流; 系统所用的 LLM 提示词逐字载录于 section B。本版本的经验研究尚属初步, 受控基准留待今后工作。

1.5 结构

Section 2 自顶向下概述 ARI: 编排器、十四个技能、流水线 DAG 与设计原则。Sections 3 and 4 是两个技术核心, 各自含有其形式化定义;section 1.2 的主张验证主线在二者间展开。Section 5 报告初步观察;section 6 定位本工作;section 7 汇集开放问题。

2 系统概览

2.1 完整堆栈

Figure 1 将 v0.9.0 的 ARI 完整堆栈画为五层; 本节自顶向底逐层讲解。系统分为一个驱动器与多个工具提供方: 驱动器 (`ari-core`) 拥有编排器、搜索与智能体引擎、谱系遍历器、检查点序列化器、成本追踪器与配置加载器, 而每一项领域能力——编写代码、运行基准、评审草稿——都驻留在一个技能之中。

入口。 一次运行从命令行界面 (CLI)、公共 Python 界面 (软件开发工具包, SDK) 或可视化服务器启动——后者是一个 HTTP 仪表盘, 向浏览器暴露实时搜索状态、按检查点的文件以及启动控制。所有路径都汇聚到同一个编排器入口 (徽标 1); 交互式使用并不会走第二条代码路径。

引擎层。 最佳优先树搜索 (best-first tree search, BFTS) 引擎维护由研究步骤节点构成的树, 并决定下一步运行哪个节点以及如何扩展它 (section 3) ——这是一种「在候选方案上搜索」的框架, ARI 与 AIDE 共享此构想 [5]。每次扩展都把一个节点交给 `AgentLoop` ———一个 Reason-and-Act(ReAct) 智能体 [1], 它将自由文本推理与工具调用交织进行, 直到该节点的工作完成: 引擎提供方向, 智能体提供执行——即一条迭代循环 (徽标 2)。位于循环旁的谱系遍历器是被咨询而非被驱动的: 它跨检查点攀爬父指针, 使派生运行得以继承其祖先的想法池与上下文 (section 3.7)。

MCP 分派。 每次工具调用都要穿过分派层 (徽标 3)。模型上下文协议 (Model Context Protocol, MCP) 是 Anthropic 标准化的 JSON-RPC-over-stdio 接口, 用于将工具式能力暴露给语言模型智能体循环;ARI 以此作为 `ari-core` 与每个技能之间的统一接缝。每个技能都是一个独立的 Python 包, 其服务器以池化子进程运行, 且当技能自带解释器时便在其自身解释器下运行。技能通过一条配置项接入, 该项指明其包路径以及其工具被暴露的流水线阶段; 工具管理器按当前活动阶段过滤工具, 因此探索节点无法调用论文撰写工具 (section 3.6)。

当前在树的十四个技能包为:`benchmark`、`coding`、`evaluator`、`hpc`、`idea`、`memory`、`orchestrator`、`paper`、`paper-re`、`plot`、`replicate`、`transform`、`vlm` 与 `web`。fig. 1 中的注册表按生命周期阶段将其工具面分组: 生成 (idea、coding)、评估 (evaluator、review)、撰写与复现 (paper、replicate、paper-re), 以及基础设施与输入输出——其中 `hpc` 技能是通往集

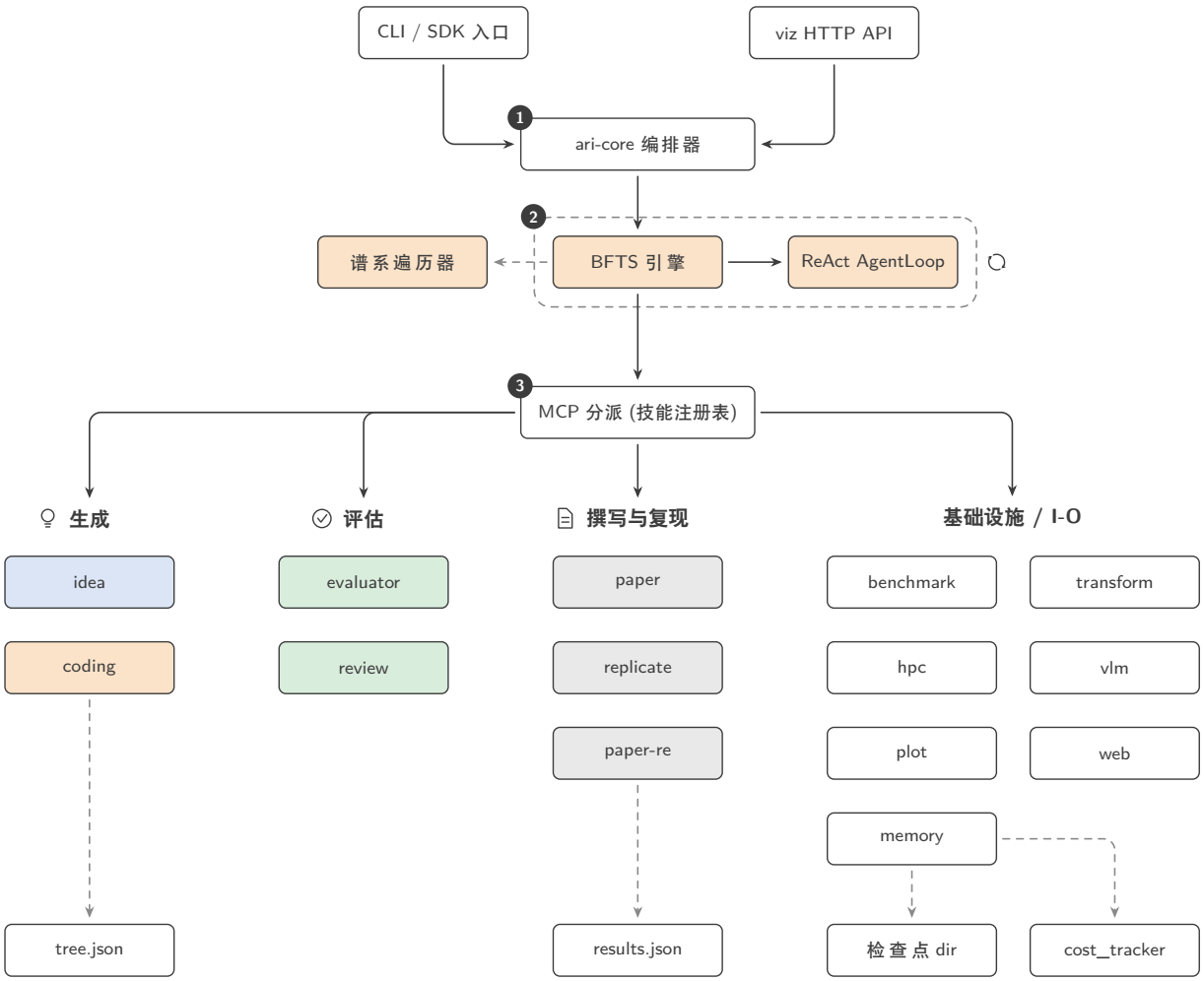

图 1: v0.9.0 时 ARI 的完整堆栈, 分为五层: 入口; `ari-core` 编排器 (徽标 1); 引擎层——BFTS 引擎与 ReAct `AgentLoop` 在此构成迭代循环 (徽标 2), 谱系遍历器位于其旁;MCP 分派枢纽 (徽标 3), 向技能注册表的四个生命周期列扇出; 以及检查点持久化。实线为控制, 虚线为次级流 (谱系查询、持久化)。

群批处理调度器的接缝, 使实验可作为派发作业运行。有两个方框值得一提:*review* 命名的是一个工具面而非一个包 (草稿评审工具由 evaluator 与 paper 技能托管;section 4.5), 而 `orchestrator` 技能经 MCP 暴露运行管理, 使 ARI 自身可作为工具被外部智能体驱动。

编排器分派调用, 但自身不执行任何领域工作。这种分离至关重要, 因为技能可被增删或替换而不必触动搜索算法:BFTS 引擎并不知道哪些技能参与运行。

最底层是持久化: 一次运行产生的每一个字节都落入单一检查点目录, 与序列化的树及成本账本并列 (section 2.5)。fig. 2 中较小的图是本报告其余部分使用的「运营」视图 (仅控制总线与状态边);fig. 1 则是「分层」视图 (驱动器 → 引擎 → MCP → 技能 → 持久化)。

2.2 流水线 DAG

一次运行可归约为一条四工位的流水线 (fig. 3)。idea 工位选取或生成研究问题;exploration 工位在 BFTS 下扩展节点; paper 工位将最佳谱系汇编为草稿;review 工位以评分细则评估草稿。该图是一个 DAG 加上一条由 review 回到 paper 的反向边: 仅此边可能引入非单调性 (改稿可能恶化某一轴), 而收敛标准 (section 4.5) 则限定反向边的数量。流水线由编排器中的单一运行器驱动; 一个科学数据组装器汇集撰写器所需的按节点产物 (section 4.2)。

撰写与评审带隐藏着一条固定的门链, 草稿在定稿前必须通过它。撰写之后, 草稿的论断经稳定标识符与已记录的证据相链接; 一个确定性的论断—证据门 (无语言模型) 在容差范围内由已记录的结果重新推导出每一个所报告的数字; 一个互补的语义评审标记过度声称并喂入改稿循环; 改稿之后, 链接、评审与门会再运行一次, 作为最终核查——在严格策略下, 只要存在任何客观不符, 该核查便阻断定稿 (section 4.3)。

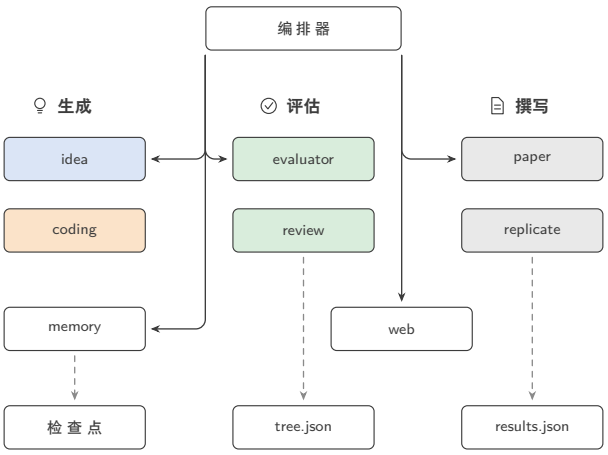

图 2: 运营视图。编排器经两条分派总线向三个生命周期列(填充编码阶段)以及无阶段的支撑技能扇出。虚线为状态:每个生产组持久化至底行的检查点产物。

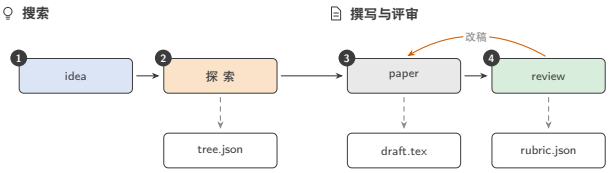

图 3: 流水线 DAG: 两个带标题 *search* 与 *write & review* 之下的四个工位 (徽标 1–4)。每个工位将其产物持久化至检查点 (虚线)。唯一着重标示的边是 *改稿*, 即重写草稿并重跑评审的反向边。

阻断门是经一层薄技能包装器抵达的 core 侧代码, 因此替换某个技能并不能削弱它; 阻断仅保留给客观谬误, 而主观评审结论引导改稿器, 却绝不对发布设门。该门链将 Story2Proposal 的契约穿线式生成以执行落地的形式实现 [8]。

2.3 BFTS 控制流

Figure 4 端到端追踪单次 BFTS 步骤。其实现在 section 3 中铺陈; 该图固定本报告其余部分使用的词汇 (前沿、选择、回退、expand、prune、停滞), 并展示两个 LLM 主导工位背后的提示文件 (`bfts_select.md`、`bfts_expand.md`); 第三个提示 `bfts_expand_select.md` 在完整循环中驱动「该扩展哪个已完成节点」的抉择 (section 3.3)。脊柱下方的两个提示框在 v0.9.0 中为新增, 实现了对计算证据的谱系链接: 覆盖度提示将扩展目标导向已持有所需测量值的节点, 而谱系提示告知新子节点它持有哪些继承而来的测量文件。这些提示仅携带名称, 绝不携带数值, 因此分支隔离得以保持 (section 3.8)。

2.4 paper-re 组件视图

Figure 5 是 `paper-re` 的第二级视图, 该技能承载 PaperBench 协议的复现路径 [9]。在此层 ARI 的贡献小而具体: 一层薄适配器包裹每个智能体工具以限定其输出大小, 并将 vendor 化求解器的硬编码路径重定向到 ARI 的按节点工作空间; 其余一切皆从 PaperBench 上游逐字流过, 这使 ARI 与上游的复现任务框架及评分语义保持一致。Section 4.6 详述 bridge 所驱动的三阶段 rollout–reproduce–grade 契约。

2.5 检查点作用域与设计原则

ARI 围绕五条设计原则 (P1–P5) 构建。最具约束力者为 **P2: 确定性**。每一步随机化均记录其种子; 每次 LLM 调用均记录其模型身份、提示与输出; 每次工具调用都以其节点身份标记 (一道写时复制护栏), 以确保其写入落在所属节点的工作树之内。重跑一个检查点必须逐字节再现相同的产物, 除非字段被显式标记为时钟相关或模型端非确定。

其余原则简述如下:

- **P1 单一产物树**: 一次运行的每一个输出都位于 `$ARI_CHECKPOINT_DIR` 之下。不外溢至 `$HOME` 或 `/tmp`。
- **P3 小组件**: 每个技能都是独立的包; 替换某个技能不需重建 `ari-core`。
- **P4 显式谱系**: 每个节点与每个检查点都记录其父级, 因此派生实验只须重算其差量。
- **P5 成本透明**: 成本追踪器为每次模型调用附加美元成本, 使用户得以为一次运行设界; 技能子进程追加到同一条运行级账本。

检查点序列化器写出三个顶层文件: 完整搜索树写入 `tree.json`, 供论文流水线消费的轻量按节点导出写入 `nodes_tree.json`, 运行末聚合写入 `results.json`。跨检查点的谱系遍历经由每个检查点记录在其运行元数据中的父指针, 自子节点向祖先攀爬。

Figure 6 描绘检查点的磁盘形状: 左侧为共享运行级文件, 右侧为按节点工作树。一个节点的工作树起初是其父级工作树的副本, 因此每棵树都是该步骤上实验的自包含快照, 而为证据组装奠基的自报告 (section 4.2) 就紧挨着它所描述的产物。该形状即契约: 任何把 ARI 输出当作输入的工具——后续运行、回归检查、论文草稿——都指向单一这样的目录。

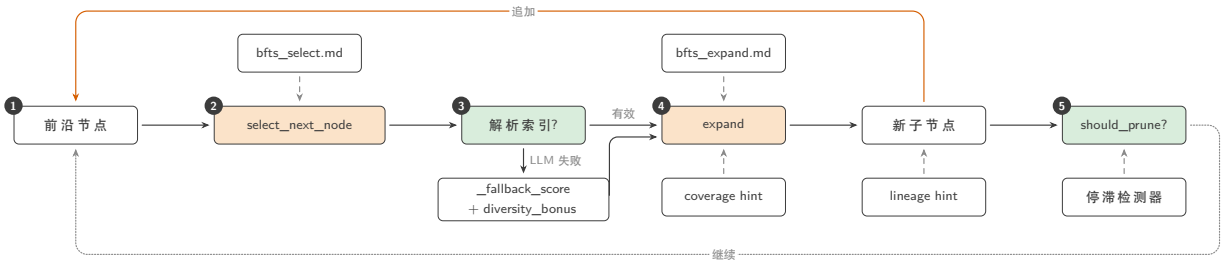

图 4: BFTS 控制流: 一步被画为由五个编号工位组成的脊柱。两个 LLM 主导的工位各自查询其专属的提示文件 (虚线, 上方); 当解析门拒绝返回的索引时, 选择回退到确定性的 `_fallback_score`(科学分数加 `diversity_bonus`)。脊柱下方:v0.9.0 的谱系链接提示注入扩展, 停滞检测器注入剪枝门。着重标示的边将子节点追加到前沿; 点线边缘路径延续至下一次迭代。

3 探索算法

探索阶段是一次运行的实验核心: 给定一个选定的构想及其所拥有的指标契约——运行级的可证伪主张声明, 每条主张都点名了哪些确切的测量名才算作其证据 (section 4.3)——探索必须把有限的节点预算转化为能让这些主张可被评估的测量。ARI 把该阶段组织为对研究步骤进行的最佳优先树搜索 (best-first tree search,BFTS)。其骨架源自 AIDE, 后者把 LLM 驱动的工程刻画为在候选解的树上、在硬编码的草稿/调试/改进策略与单一标量目标下的搜索 [5];ARI 用 LLM 主导、由确定性回退兜底的选择取代该策略 (section 3.3), 用契约导出的覆盖度与谱系提示来引导扩展 (sections 3.4 and 3.8) ——这是对贪心单目标策略所报告的局部最优平台期 [5] 的一种结构性回应——并把每个节点作为受保护的「推理-行动」(Reason-and-Act,ReAct) 智能体执行, 其义务与反馈都来自同一份契约 (section 3.6)。收尾各小节覆盖搜索运行所依托的底层: 平台探测 (section 3.9)、研究记忆 (section 3.10), 以及为根节点播种的构想阶段 (section 3.11)。

3.1 形式化

探索循环维护一棵树 $\mathcal{T} = (\mathcal{N}, E)$, 其中每个节点 $n \in \mathcal{N}$ 代表一次研究步骤: 构想修订、代码改动、基准运行、分析等。节点携带

- 离散状态 $s(n) \in \{\text{open}, \text{eval}, \text{done}, \text{failed}\}$,
- 取自 `NodeLabel` 枚举的类别标签,
- 用于按轴评估指标的自由形式 dict `node.metrics`, 含独立标量 `_scientific_score`(取值范围 $[0, 1]$),
- 标志节点是否携带实测值 (而非临时文本) 的布尔 `has_real_data`, 以及
- 选择提示与剪枝谓词使用的簿记字段 `depth`。ARI 不重试失败节点, 因此不会暴露任何 `retry_count` 信号。

Figure 7 展示了一棵这样的小树。将按轴信号汇总为 scientific score 的工作委托给任何实现 `Evaluator` 协议的对象;ARI 不固定线性组合。该协议是 BFTS 与下游评分细则评估之间唯一的接缝, 保持搜索引擎对如何将轴归约为标量保持中立。

Run-loop 状态. 外层循环每次迭代变换以下元组:

$$S = (\mathcal{F}, \mathcal{P}, e, H),$$

其中 $\mathcal{F} \subseteq \mathcal{N}$ 为可扩展的完成节点集合 (done / failed),$\mathcal{P} \subseteq \mathcal{N}$ 为处于 open 等待执行的节点集合,$e : \mathcal{N} \to \mathbb{N}$ 记录每个前沿节点被扩展过几次,$H = (l_1, \ldots, l_t)$ 是已执行标签的滑动窗 (长度上限 $W = 20$, 详见 section 3.3)。初始状态为 $S_0 = (\emptyset, \{n_{\text{root}}\}, \mathbf{0}, ())$。

剪枝谓词. 硬性探索预算由如下谓词表达:

$$\Pi(n; |\mathcal{N}|) = \mathbb{1}[|\mathcal{N}| \geq B_N] \vee \mathbb{1}[\text{depth}(n) \geq B_D] \vee \sigma(n), \quad (1)$$

其中 B_N = `config.max_total_nodes`,B_D = `config.max_depth`,sterile 谓词

$$\sigma(n) = \mathbb{1}[\,|\Delta_n| = 0\,] \quad (2)$$

(Δ_n 为 n 新增、修改或删除的文件集合) 当节点在 agent loop 结束时相对父节点未写出任何新工件时为真。run loop 计算该文件差分, 并把结果作为布尔 `_sterile` 标志记录到节点上;`should_prune` 只读取该标志与两个上限。调用方每次迭代传入活动的 $|\mathcal{N}|$, 因此节点数有唯一真值源。步数与成本预算由调用侧的成本追踪器单独强制, 而不在 BFTS 引擎内。

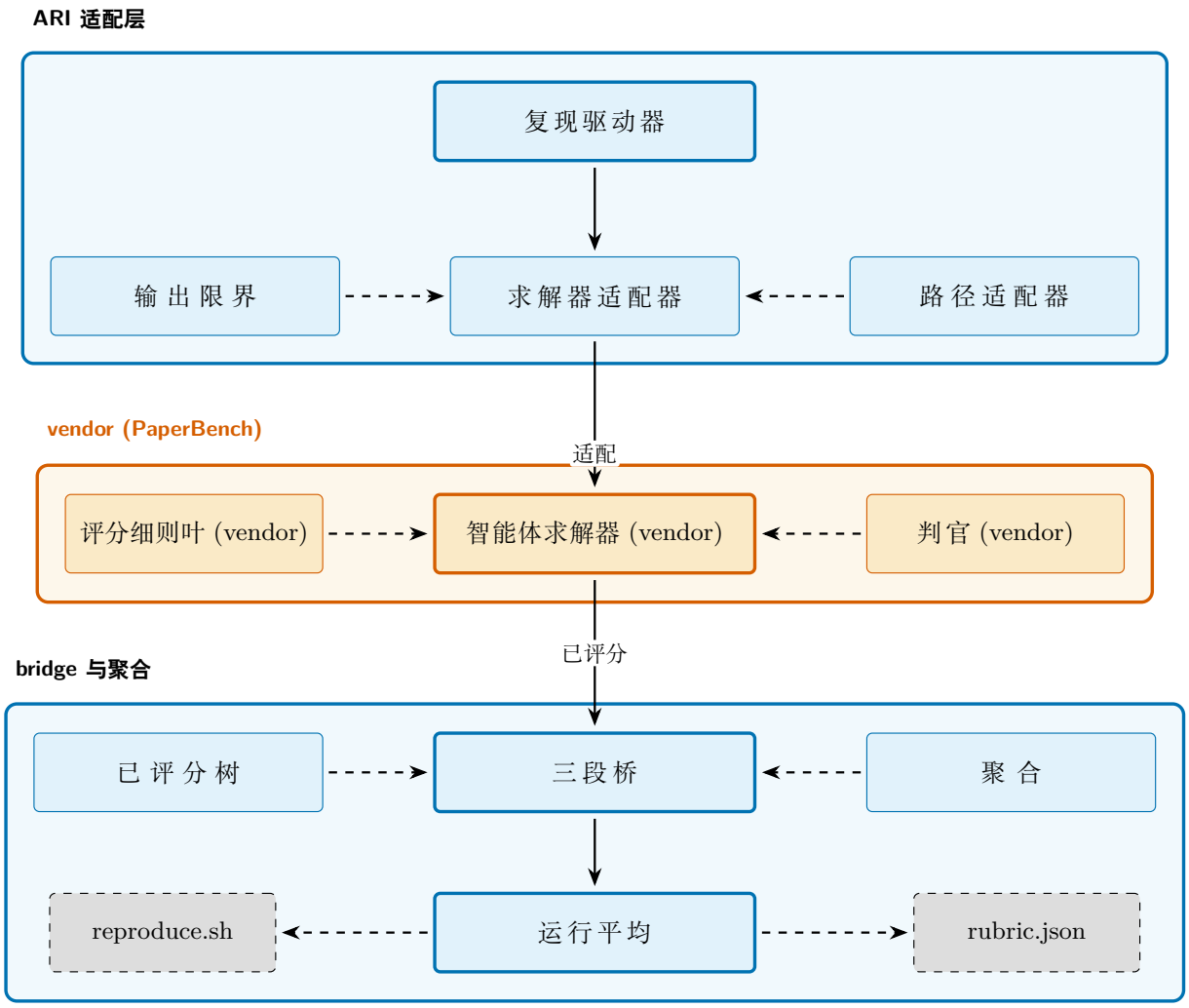

图 5: `paper-re` 技能的组件, 分三层。顶行为 ARI 的适配层 (复现驱动器、求解器适配器、输出限界、路径适配器); 中行为 vendor 化的 PaperBench 包 (智能体求解器、评分细则叶、判官); 底行为 bridge, 它驱动三个协议阶段, 并把按 submission 的已评分树——对重复运行取平均——折叠成单一 `rubric.json` 得分, 加上 ARI 侧的 `reproduce.sh`。详见 section 4.6。

Retire 谓词. 在 Π 之上,run-loop 还应用更软的前沿 *retire* 谓词 ρ, 在父节点 f 不再是最高产线索时将其从 $\mathcal{F}$ 中移除:

$$\rho(f, c) \;=\; \underbrace{\mathbb{1}[r(c) > r(f)]}_{\text{规则 A}} \;\vee\; \underbrace{\mathbb{1}[e(f) \geq E_{\max}]}_{\text{规则 B}}, \qquad (3)$$

其中 $E_{\max} =$ `config.max_expansions_per_node`(默认 4)。规则 A 在子节点 c 一旦超越 f 时退役 f; 规则 B 限制每个父节点的预算, 从而展开搜索面。ρ 不删除节点, 其历史仍保留在 $\mathcal{T}$ 中, 仅撤销其继续被扩展的资格。

外层循环单次迭代的完整阶段分解 (内层扩展子循环、并行 ReAct 批次、执行后 retire) 见 fig. 8; 显式伪代码见 section 3.2 的 algorithm 1。

3.2 Run-loop 算法

Algorithm 1 把该迭代写成显式伪代码; fig. 4 把同一步骤呈现为带提示文件的控制流主线。内层 WHILE 循环每次填入一个空 worker 槽位 (因此 worker 必在下一次提出扩展前开始运行); 外层块随后通过反复调用 `select_next_node`(每次一个节点)组装批次, 直至达到 worker 上限, 再并行执行该批次。执行后块 (a) 把已执行标签追加到 H 以使多样性奖励反映真实执行, (b) 应用 ρ 的规则 A 与 B。

3.3 节点选择 (LLM 主导)

ARI 有意不将节点排序归约为闭式标量效用——这一设计抉择正是其 BFTS 与 AIDE 硬编码贪心策略 [5] 最大的分野。相反,`select_next_node`(「下一智能体步该运行哪个 pending 节点」) 与 `select_best_to_expand`(「哪个完成节点最值得扩

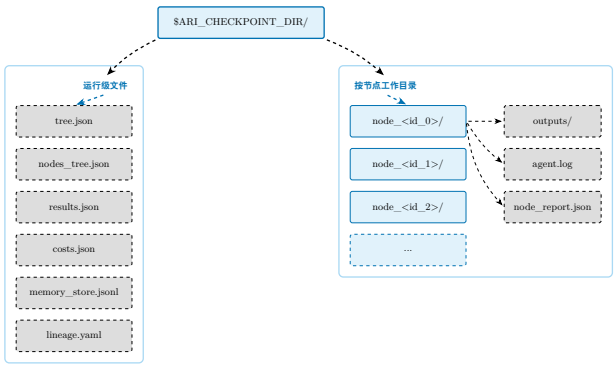

图 6: 检查点目录布局。运行级文件 (左带) 由编排器写入; 按节点工作树 (右带) 保存一次 BFTS 扩展内产生的产物, 包括每个节点留下的结构化自报告。复现一次运行恰好就是「针对此目录重跑」, 而该布局保证不需要任何其他状态。

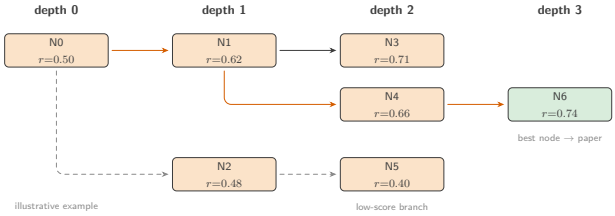

图 7: 一棵小型 BFTS 探索树 (合成、示意)。列为搜索深度; 每个方框是一个节点, 标注其标量奖励 $r(\cdot)$。加重的边描出最佳节点回溯到根的谱系——这正是论文阶段所消费的链条 (section 4.2); 虚线边标记了一条选择策略停止选取的低分支, 其历史仍保留在树上。

展」) 都把候选描述列表交给 LLM, 并解析返回的单个 0 起始索引——每次调用恰好选出一个节点。每个节点 n 的候选描述 (fig. 9) 形如:

```
[i] id=..., depth=..., label=...,
has_real_data=..., metrics={...},
summary=...
```

运行选择侧 (`select_next_node`) 会对当前获得奖励的节点在行尾额外追加 `diversity_bonus=+0.05`;`select_best_to_expand` 不追加。消费该列表的提示分别于 section B.3(选择) 与 section B.2(扩展目标) 逐字载录。提示中列出的选择标准:`has_real_data=True` 且 metrics 强者为高价值;metrics 整体加权 (多目标); 深而成果出色的节点值得继续; `diversity_bonus` 仅作软性平手解决项处理。`label` 字段与展示给 `select_best_to_expand` 的信息一致, 并且不存在误导性的「偏好低 retry」标准。

多样性奖励

多样性奖励 `BFTS.diversity_bonus` 以 `_recent_label_history`(由 `record_run` 填充的滑

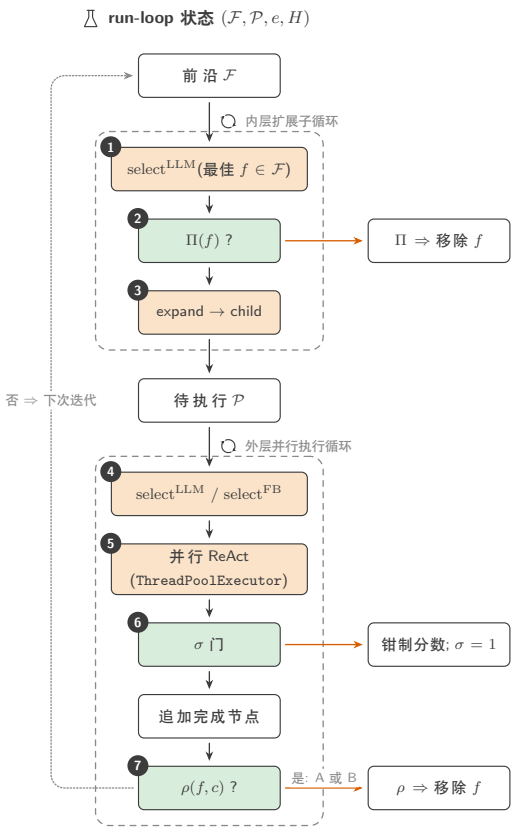

图 8: BFTS 单次外层迭代的状态机视图 (自上而下的单一流程图)。步骤 1–3 为内层扩展子循环; 步骤 4–7 为外层并行执行循环。黄色菱形为谓词门 (Π, σ, ρ); 右侧红色虚线框为各谓词触发的副作用 (前沿移除 / 分数夹紧 / retire)。图中未画出的辅助状态: 每父节点的扩展计数器 $e(\cdot)$ 在步骤 3 自增, 在步骤 7 的 ρ 中被使用; 标签历史 H(窗口 W) 在步骤 5 由 `record_run` 追加, 在步骤 4 用于计算多样性奖励 $\mathrm{div}(\cdot)$。可与 fig. 4(将相同循环呈现为带 LLM 提示的细粒度控制流主线) 对比。

动窗) 跟踪近期标签。对于其标签在该窗中相对最常见标签为 欠表达的节点, 函数返回:

$$\mathrm{div}(n) = \begin{cases} +0.05 & \mathrm{cnt}(n) \le \frac{1}{2}\,\mathrm{cnt}_{\max}, \\ 0 & \text{否则}, \end{cases} \tag{4}$$

其中 $\mathrm{cnt}(n)$ 为窗内 $\mathrm{label}(n)$ 的出现次数, $\mathrm{cnt}_{\max} = \max_{\ell} \mathrm{cnt}(\ell)$。0.05 这个常数有意设小:scientific score 仍主导排序, 奖励仅打破近似平手。

LLM 回退

若 LLM 返回无法解析的索引,`select_next_node` 通过 `_select_fallback` 辅助函数 (按 `_fallback_score` 式对候选排序) 回退至确定性排序。默认评分式为

$$\mathrm{fb}(n) \;=\; r(n) + \mathrm{div}(n), \tag{5}$$

其中 $r(n) \equiv$ `_scientific_score`(n) 是节点的标量奖励 (section A)。`_select_fallback` 在存在

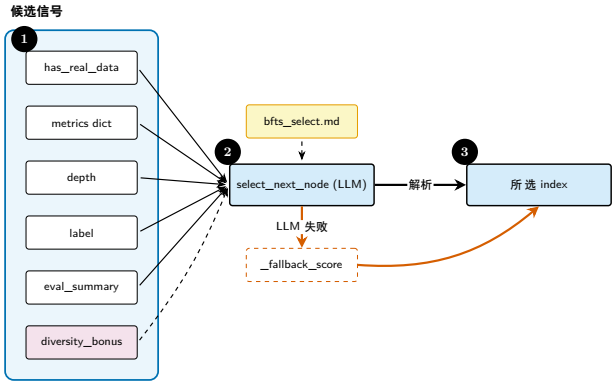

图 9: 交给 LLM 用以挑选下一节点的选择信号。输入被拼接成结构化候选描述; 软性的 `diversity_bonus` 仅作平手解决, 不作主信号。

`has_real_data=True` 节点时优先选取它们, 然后返回 fb(·) 最大的候选。这保证即使 LLM 调用退化, 循环也始终向前推进。

评估层的配置

sections 3.1 and 3.3 中出现的 4 个评估面可独立配置; 默认值复刻上式, 因此未修改的配置等同于 no-op。

合成公式. 由 `evaluator.composite` 选择。$\{a_k\} \to$ `_scientific_score` 的归约方式之一: 加权调和平均 (默认)、加权算术平均、瓶颈式 `weighted_min`、加权几何平均。调和与几何平均对零值轴施加下界 $\epsilon = 0.01$, 以保证分母有限。

前沿评分策略. 由 `bfts.frontier_score` 选择。eq. (5) 对应 `scientific_plus_diversity` 情形; 其余可选 `scientific_only`(去除 div); `depth_penalized` 在 $\mathrm{div}(n)$ 之上叠加 $-\lambda\,\mathrm{depth}(n)$, 其中 $\lambda =$`depth_penalty_lambda`; `ucb_like` 在 $\mathrm{div}(n)$ 之上叠加 $c_{\mathrm{ucb}}\sqrt{\log N/(e(n)+1)}$, 其中 $c_{\mathrm{ucb}} =$`ucb_c`、$N = \sum_{n'} e(n') + |\mathcal{F}|$。

评估轴集合. 由 `evaluator.axis_mode` 选择。`dynamic`(默认) 由通用 5 轴 floor 加上有效 rubric 与 `idea.json` 的 plan-keyword 构建轴集;`legacy` 固定为 5 个标准轴;`custom` 把显式的 $\{(\mathrm{name}, \mathrm{description}, w)\}$ 列表逐字喂给 judge LLM。

选择 prompt. 由 `bfts.select_prompt` 与 `bfts.expand_select_prompt` 设置: 一对 `Filesystem PromptLoader` 键让用户无需改源码即可为 algorithm 1 中两个 $\mathrm{select}^{\mathrm{LLM}}$ 调用换入替代模板。选择模板须接受占位符 `experiment_goal`、`memory_context` 与 `candidates`; 扩展目标模板接受 `experiment_goal` 与 `candidates`。

3.4 扩展 (每次调用 1 个子节点)

`expand` 函数每次调用至多生成一个新子节点 (不预先成批)。同一父节点希望更多子节点的调用方反复调用 `expand`, 通过 `existing_children` 参数把已生成的子节点串入, 使 LLM 得以避免提议重复方向。

各新子的标签由 LLM 判断而非硬编码模板决定; 扩展提示 (逐字载录于 section B.1) 接收:

- 父节点状态 (`succeeded` / `failed/no-real-data`) 与 `_scientific_score`;
- 父节点结构化的 `node_report` (`delta_vs_parent` / `concerns` / `hints`, 由 node-report 构建器生成) 若存在, 以便规划者瞄准具体弱点;
- 同深度的兄弟得分, 以及自根至父的祖先得分;
- 整树多样性指标 (目前已见的唯一标签、深度分布); 以及
- 此父节点已生成的子节点标签, 避免重复提议。

子节点以 open 创建, 交给智能体循环执行 (section 3.6), 所有轴填齐后变为 done; 智能体循环抛异常时变为 failed。

覆盖度引导. 扩展目标选择器本质上是一个跨分支调度器——它看得到每个分支的得分、metrics 与摘要——但单凭得分并不能反映运行所声明的主张: 在一次真实运行中, 一棵十节点的树产出了同一头条实验的十个变体, 而声明的机制类主张却没有得到任何专门实验。因此, 交给 `select_best_to_expand` 的目标文本被一个由指标契约导出的运行级*主张覆盖度*块所扩充: 哪些已声明主张已在运行某处获得支撑测量、哪些仍未覆盖, 使得「能覆盖一个尚未覆盖的主张」可以参与决定应扩展哪个节点。一条主张所需的测量名即其*契约证据名*。覆盖度在两个层面保守地计算: 仅当一条主张所需名称的*多数*已被发出时, 才算其被覆盖——故意比最终门的「任一名称」规则更严格, 因为对引导而言, 一个偶然共享的通用名会永久压制掉那个专门实验, 而误判为未覆盖只是可能多做一次冗余测量——且仅有被评估器认定携带实测数据的节点才作出贡献, 与门保持一致, 使得有缺陷分支仅有的测量*名称*无法告诉兄弟「已覆盖」。同一状态按节点重算后也送达每个执行中的智能体 (section 3.6); 那些把*已计算证据*沿单一谱系串联的配套提示, 则是 section 3.8 的主题。

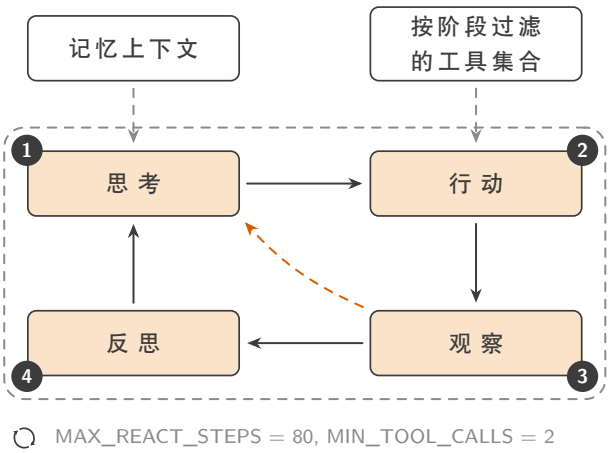

图 10: ReAct 循环。实线为主循环; 虚线反向边表示当观测已闭合开放问题时的提前退出。

3.5 剪枝与终止

`should_prune` 实现剪枝谓词 Π(eq. (1)): 总节点上限、深度上限 (`max_depth` 配置字段) 与 sterile 标志。前沿退役谓词 ρ(eq. (3)) 在每个节点完成后施加; 它不删除节点, 其历史仍保留在 $\mathcal{T}$ 中, 但前沿池会收缩, 使搜索分散而非无限挖掘同一父节点。

下列任一条件满足即停止:

- 全局 `max_total_nodes` 上限被触及;
- 前沿中每个节点都触发 Π(深度上限、sterile 或预算耗尽) 且 pending 池为空;
- 成本追踪器在调用侧强制的步数 / 成本预算耗尽;
- 停滞检测器 `detect_stagnation` 观察到 `_scientific_score` 的运行最大值在滑动窗内不再改善;
- 谱系判断模块的 `LineageDecision` 显式终止该分支 (参见 section B.4)。

这些当中实际最常发生哪一种, 是一个经验问题, 留待冻结检查点就位后由 section 5 回答; 本报告不主张其分布。

3.6 ReAct 驱动器

每个节点的扩展在 `AgentLoop` 类中运行一个 ReAct 智能体循环 [1](fig. 10): 模型在其累积上下文上推理、发起工具调用、观测结构化结果, 然后重复。最大步数由 `MAX_REACT_STEPS = 80` 控制, 可按实例通过 `max_react_steps` 关键字覆盖。另一保护 `MIN_TOOL_CALLS = 2` 防止平凡过短的轨迹。智能体的系统提示词逐字载录于 section B.6。

智能体可用工具集合按阶段由工具管理器过滤; 例如探索阶段屏蔽论文撰写工具, 使 BFTS 扩展不会捷径地变成草稿。一小部分 MCP 工具为 `ari-core` 自身预留 (memory CoW 操作) 并对 LLM 隐藏; 这保证模型无法设置任意 node id 来绕过记忆技能的写时复制校验。

节点起始的义务

节点并非从空白提示开始。在首次模型调用之前, 循环注入确定性上下文:section 3.10 的工作上下文——实验的固定核心加上其祖先的结论——以及, 一旦运行的指标契约存在, 一条 契约义务消息, 以领域中立的措辞陈述节点必须做什么, 才能让最终的主张在科学上成立而不仅是貌似合理: 在头条数字所用的问题规模上, 对照一个独立参照验证正确性, 并把残差作为一次测量记录下来; 对任何用作归一化分母的上限都要测量而绝不硬编码; 为每个上限与正确性检查标注来历, 以便门可以确认它们确实运行过; 声明头条指标如何从其原始操作数重新计算得出; 并在确切声明的名称下发出那些能让每条声明主张可被评估的测量——负面结果可以接受, 无证据的主张则不可。该文本不携带任何领域知识; 智能体按其领域所需履行每项职责 (为吞吐上限做微基准, 为精度上限跑一个基线, 为数值方法做一次解析检查)。

三个运行级附件搭载同一条消息: 节点的主张覆盖度状态 (section 3.4), 它要求节点优先攻克一两个它可行测量的尚未覆盖主张;section 3.8 的继承数据说明; 以及 section 3.9 的平台说明。这些运行级不变量是被钉住的: 滑动上下文窗始终保留它们——连同综述、构想生成与指标规格 (metric-spec) 工具的结果——而较旧的回合则被淘汰。在根节点处, 上下文构建时契约尚不存在, 该义务由契约铸造工具调用的处理器在循环中途注入, 因此没有任何节点对声明主张盲目运行, 也没有任何节点收到两次义务。

发出反馈

测量通过单一的结构化发出工具 (`emit_results`) 离开节点, 该工具写出节点的结果文件、对照指标契约校验, 并返回契约警告, 点名任何仍缺失的必需证据。早期运行在此处恰好暴露出一种失败形态: 警告抵达于一个工具结果之内, 而 harness 在节点的首个已完成作业之后立即强制收尾, 未满足的义务被留下没有可供行动的步骤——一个被观察到的节点结束时其 80 步中有 66 步未用。因此循环把发出时刻的反馈转化为一个可操作的回合: 当一次发出携带契约警告时, 它注入一条继续推进的提示——

缺失的测量、剩余的步数，以及现在就去运行它们并重新发出（重新发出会覆盖）、或仅当它们在本节点不可行时才作结的指令——并把节点置于一个契约保持态，该态抑制下文所述的强制收尾保护。该保持态在步数上限前十步过期，以保留反 doom-loop 的兜底；一次无警告的发出会清除它。

强制收尾保护

循环对轨迹的两端都设防。针对过早的成功主张，节点在 `MIN_TOOL_CALLS` 次工具调用之前不得收尾，且——只要其工具集包含一个执行工具——在它执行过某物并读取其输出之前也不得收尾。针对 ReAct 智能体被记录的重复循环失败形态（原研究将很大一部分推理失败归因于反复发出近乎相同的动作 [1]），循环会强制收尾：一旦节点已有一个其输出被读取的已完成作业、至少经过五步、且无契约保持态生效——或在满足最低保护后、上限前的最后三步内——工具调用要求即被取消，使模型的下一次输出必须是其最终结构化报告。重复在工具层面也被抑制（构想生成工具在其首次成功调用后被压制），且注入的用户消息被推迟到消息块边界，使重放的对话从不把工具调用与其响应割裂。

3.7 谱系与可复现性

谱系即最佳节点祖先的链条，作为 `tree.json` 中的 `lineage` 键写入检查点。`LineageState` 数据类捕获 BFTS 用于决定是否继续的滚动统计；跨检查点的遍历器 `walk_ancestor_ckpts` 提供跨运行的父指针。构想池经 `get_idea_pool_for_ckpt` 继承，排序后的根构想选择记录在 `RootChoice` 数据类中以便事后审计（提示词：section B.5）。

可复现性依赖三项不变量。其一，每次随机化操作以节点标识符播种。其二，每次工具调用将参数与输出写入所属节点的工作目录，绝不写入父节点——由 MCP 层的 `cow_node_id` 校验强制。其三，成本账本在 `CostTracker.init` 处为每节点附加美元金额，因此给定相同提示，预算检查保持确定。

默认离线。 可复现性也规定搜索循环可以读取什么。默认下每个节点的智能体循环离线运行——Web 访问技能被相位限制到论文撰写与复现阶段，因此节点无法在搜索过程中查询随时变化的实时检索结果，每节点轨迹也就不依赖于其运行当天的 Web 状态。文献查阅仍会发生，但更早且在循环之外——在搜索开始前、构想阶段执行的一次有界综述（section 3.11）。确实需要实时信息的运行可以选择性开启（`bfts.allow_web`），在探索期间向节点智能体开放 Web 访问；此时运行会向其检查点写入一个不可复现轨迹标记，使日后的复现不会被默默当作精确。无论哪种方式，确定性契约都保持诚实：默认运行按字节可复现，而以可复现性换取实时信息的运行会把这一权衡记录在自身来历中。

3.8 面向计算型证据的谱系串联

并非运行打算支撑的每条主张都来自新的测量。有些证据是从已经存在的测量计算而来——对早先探针数据的参数拟合、对该拟合的留出验证（held-out validation）、在多个构型间基于模型的选型。事实证明，这类主张对相互独立的树节点是结构性不可达的：在一次真实运行中，以拟合或验证为目标、却从一个不携带任何测量的父节点扩展出的子节点，退化为重新运行单个探针——因为「用你已有的数据来计算」从来不是一个可见的选项：子节点手里没有数据，也没有任何信号告诉选择器哪个节点有。

弥合这一缺口的机制只使用树本来就有的父节点 → 子节点工作目录继承（子节点在父节点工作树的副本内执行；section 4.2 依赖同一事实），在扩展的两侧各加一个提示。父侧：追加到扩展选择目标的主张覆盖度块（section 3.4）获得一个谱系提示，点名其工作树持有最多契约证据名的节点，使拟合或验证型的子节点优先从已持有其输入的节点扩展。子侧：当子节点继承的目录中含有谱系测量文件时，它在起始（section 3.6 的继承数据说明）即被告知哪些文件存在、其中含有哪些确切的契约名，并被指示从这些文件计算派生证据，而非重新运行底层实验。

跨越节点边界的只有名称：文件名与测量名，从不是数值，也从不是兄弟分支的结论。这些提示与契约本身同属运行级协调元数据，因此 section 3.10 的分支隔离原样保持——有缺陷分支的数字仍然无法泄漏进兄弟分支的推理——同时计算型证据链（先拟合、再验证、后选型）得以沿单一谱系执行，而不是塌缩回反复的探针运行。

3.9 平台探测

一个研究计划在对其科学判断出错之前，可能先对其平台判断出错。在一次真实运行中，声明的主张需要硬件计数器剖面，而所假定的剖析工具在实验运行所在的计算分区上被可验证地证明缺失：那些主张永久不可满足，最终门——正确地——因为没有任何节点能产出的证据而拦下了论文。因此平台可行性必须被测量而非假定——而且只测量一次，在下游任何东西承诺一套测量词汇之前。

当运行配置了批处理分区时，运行开始会发起一次性的平台探测：一个短小的调度作业，在计算分区自身上执行，按名检查一小份可配置的测量工具列表（剖析器、硬件计数器工具及类似工具）的可用性，并记录机器架构。结果被缓存为检查点工件（`platform_capabilities.json`），且在运行内不再重新探测。该探测在设计上尽力而为：未配置分区、缺少调度器，或排队等待超过其超时，都退化为一次什么都不写的已记录跳过——探测失败会让每个消费方都不受约束，且不改变下游任何东西。

随后，探测到的事实被中继——作为数据，而绝非内置硬件知识——到三个本会出现平台盲计划的层面。在构想时它们被折入研究主题，使构想阶段（section 3.11）只规划平台能采取的测量：可行性在源头就被固定，而非在下游打补丁。在契约时，从选定构想的计划导出指标契约的主张提取器把它们作为一条已验证的能力说明接收，使契约绝不声明依赖于缺失工具的证据——这一点尤为重要，因为契约一旦铸造便被冻结（section 4.3）。在节点时，被钉住的义务消息（section 3.6）携带一条平台说明，列出可验证缺失的工具，使执行中的智能体不必把其 ReAct 步数浪费在以艰难方式发现缺失命令上。关于哪些工具值得探测的知识存在于 HPC 技能的可配置列表里；核心编排器与门保持无领域，仅中继被测量到的内容。

3.10 研究记忆

节点并非孤立运行：每个节点继承其祖先所确立的内容，反过来又为其后代与论文撰写器留下记录。ARI 把这一记录保存在每次运行的研究记忆中，它遵循自由形式日志无法满足的两项要求。其一，分支隔离：节点只能读取自身祖先的记忆，绝不读取兄弟或无关检查点的记忆，因此两条并行线索无法相互污染——与 section 3.7 中定义谱系的祖先作用域谓词相同。其二，可基底性：由于记忆最终支撑论文，条目必须能指向结果背后的证据，而非仅复述结果，从而一个主张可回溯到支撑它的产物。

作为证据索引的记忆

节点产出内容的唯一真值来源是其结构化自评报告——节点完成时写出的 `node_report.json`（section 4.2 详述其字段）。记忆条目不复制这些字段。它携带一段简短可检索文本、一个种类、一个指回原始节点报告的指针，以及对结果所依赖的特定产物——日志、源文件、输出文件——的内容哈希引用。因此记忆条目是对证据的索引，而非证据本身：它所引用的产物可随时重新哈希，以确认其仍存在且仍匹配，这一完整性检查把每个引用归类为已验证、缺失或不匹配。种类取自固定词汇——观测、实验结果、失败案例、过程、反思、产物摘要、论文主张、可复现性事件——使读者能精确索取所需记录：比如某条分支上的失败，或仅由产物支撑的结果。

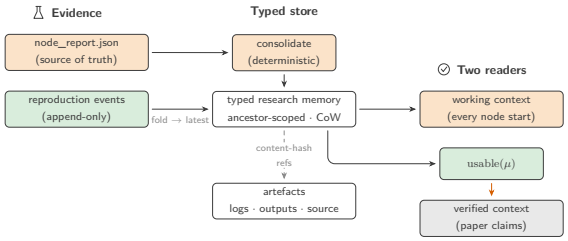

图 11： 可验证研究记忆。对每个节点真值来源 `node_report.json` 的确定性整合，填充一个按内容哈希索引产物（日志、输出、源码）的带类型存储。复现事件被追加并折叠为最新状态，准入谓词 usable(μ)(eq. (6)) 控制哪些记录进入已验证上下文。同一存储供养两个读者：在下一节点起始注入的工作上下文（探索的继承路径），以及为论文定量主张提供依据的已验证上下文（section 4.2）。无据的反思可引导探索，但绝不引导论文正文。

节点终端的整合

节点完成时，一个确定性的整合步骤读取其节点报告并发出带类型条目：为产生测量的节点发出实验结果条目，附带对其所测产物的来历引用；为失败的节点发出失败案例条目。该步骤不调用语言模型，因此同一节点报告总是整合为相同条目——整合是已记录状态的纯函数，重跑检查点会再现它所产生的记忆。它是循环钩子而非智能体动作：搜索智能体从不会被要求「保存到记忆」，实际上智能体也不会自行去取用回忆工具。循环所依赖的记忆读取由循环自身确定性地完成——正是这一点让记忆能承载来历，而不必把非确定性的检索放在关键路径上（section 7.1）。

两个读者：工作上下文与已验证上下文

该存储供养两个消费者（fig. 11），二者之别正是设计的核心。在每个节点起始，循环注入工作上下文：实验的固定核心——目标、目标度量、硬件——以及其祖先所记录的结论。于是子节点不是从聚合文本转储开始，而是从谱系已确立的内容开始，确定性地且限定于自身分支。在论文阶段，流水线从最佳节点的谱系构建已验证上下文：它把每个结果的可复现性事件折叠为最新状态，并对条目排序，使可复现的结果高于仅由产物支撑者，后者又高于无据的反思。条目 μ 仅当其由产物支撑且未被复现尝试反驳时，才可作

为定量论文主张:

$$\text{usable}(\mu) \;=\; \text{grounded}(\mu) \;\wedge\; \big(\text{repro}(\mu) \neq \text{failed}\big). \tag{6}$$

无据的反思仍可通过工作上下文引导搜索, 但 eq. (6) 把它挡在论文正文之外: 撰写器的定量主张只依赖于可作为主张、优先可复现的条目。这是证据组装器 (section 4.2) 在记忆一侧的对应物; 组装器决定撰写器可读取哪些产物字节, 而已验证上下文决定哪些结果可成为主张。

作为追加事件的可复现性

由于过去条目按字节稳定——节点只在自身标识符下写入, 既有条目从不就地修改——结果的可复现性状态在日后重跑时不会被就地改写。重跑改为追加一个可复现性事件条目, 任何读者都会把某目标的事件折叠为其最新状态。复现失败的结果因此被从 eq. (6) 的主张集合中降级——并可诚实地作为局限报告——而不必改写当初把它记录为有望的历史。

3.11 构想: 为根节点播种

树的根节点并不以代码开始; 它以决定研究什么开始。根节点的智能体运行构想阶段: 一次有界文献综述, 继之以多智能体构想生成, 收敛到一个选定构想 (检查点中的 `idea.json`), 其计划产出运行的指标契约 (section 4.3)。由于综述在此处、在构想时发生, 搜索本身得以保持离线 (section 3.7): 综述从 Semantic Scholar 检索实时文献——关键词搜索辅以对引文图的两跳遍历——随后的讨论便在检索到的摘要上展开。

构想被有意设计为多智能体。单智能体、以归档为条件的头脑风暴已知会塌缩到一个狭窄的构想分布——AI Scientist 报告其构想生成「常常在不同运行甚至不同模型间产出非常相似的构想」[2] ——而 VirSci [7] 模拟一支科学家团队 (合作者选择、圆桌讨论、构想生成、新颖性投票), 并报告该多智能体系统相对单智能体执行者, 平均在与当代研究的契合度上提升 +13.8%、在潜在影响力上提升 +44.1%。

ARI 在两种保真度上封装 VirSci。默认路径在构想技能内重新实现 VirSci 的讨论流程, 在可得处直接使用 vendored VirSci 提示模板, 把检索重新基于实时综述, 并把模型调用经由 ARI 的 LLM 层路由; 候选构想携带 VirSci 风格的自评 (新颖性、可行性、清晰度), 并按新颖性加权和排序。一个选择性开启的 *VirSci-live* 模式则直接驱动 vendored VirSci 引擎本身、未经修改 (fig. 18): 一份冻结的、源自 Semantic Scholar 的快照——一个带 SPECTER2 检索索引

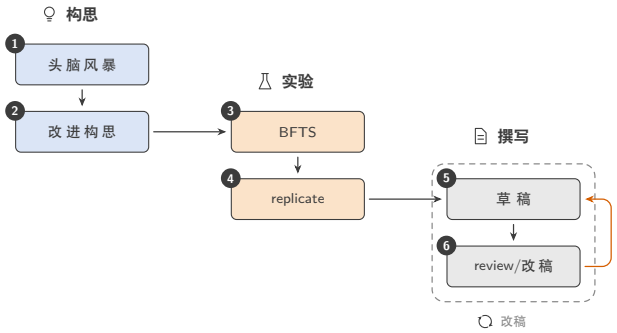

图 12: 论文生成生命周期表示为按阶段分组的列: 构思、实验、撰写 (步骤 1–6)。虚线包围标出评审/改稿对; 其反向边是生命周期中唯一的非单调转移 (section 4.5)。

与按新鲜度排序的合著者图的论文语料——被物化到检查点之下,VirSci 自己的 `select_coauthors` 在该图上组建科学家团队, 其 K 轮 `generate_idea` 讨论产出候选。因此 ARI 所依托的多智能体机制正是已发表的那个, 只是针对当前文献执行而非被转述; 冻结快照让讨论基底保持可复现, 而实时路径在任何失败时退化为重新实现的循环, 故启用它绝不会使一次运行回退。

两个运行级事实在两条路径上都被折入讨论。探测 (section 3.9) 验证的平台约束被追加到研究主题, 使每个消费该主题的提示只规划平台能采取的测量; 且在派生运行中, 祖先检查点的构想池 (section 3.7) 被只读注入, 使团队在谱系已提议内容的基础上精炼、扩展或明确转向, 而非重新推导。候选之间排序后的选择被记录以供事后审计 (section 3.7), 使运行的起点与之后每一次扩展一样可追溯。

4 论文生成流水线

4.1 流水线概述

探索终止后, 存活谱系被交付给论文生成流水线——一次运行的后半段 (fig. 12;fig. 3 将其置于运行级阶段 DAG 之中)。撰写阶段将谱系汇编为草稿, 评审阶段以评分细则对该草稿评分; 两者由改稿循环 (section 4.5) 耦合。流水线持久化三件输出:LaTeX 草稿、带理由的按叶评审, 以及本次运行所用的评分细则实例 (评分细则可能逐论文生成而非预先固定, 故保存)。

流水线的组织原则是: 草稿中的每一句定量陈述都必须可追溯到探索实际产生之物。两套机制承载该原则。确定性的*证据组装器* (section 4.2) 把撰写器可依据的*哪些产物*固定下来, 作为已记录元数据的纯函数。*主张–证据契约* (section 4.3) 则固定论文必须就这些产物*说什么*: 构思期声明的可证伪主张、可充当各主张证据的测量名, 以及一道确定性闸门——它在

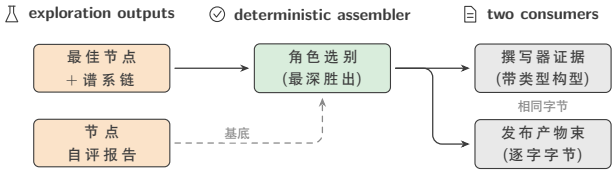

图 13: 证据组装。确定性组装器遍历最佳节点的谱系，以每个节点的结构化自评报告（内容哈希、成功自评、逐字构建/运行命令、所捕获硬件）为依据，做出一次按角色的选择，该选择以逐字节相同的内容供给两个消费者：撰写器的谱系证据与已发布产物束。每个贡献文件都由已记录元数据选出；路径冲突时最深的出现胜出，链上的删除则移除该文件。

论文得以发布之前，从已记录结果重新推导每个报告数值。其余各节——评分细则形式（section 4.4）、评审/改稿循环（section 4.5），以及 PaperBench 复现路径（sections 4.6 and 4.7）——皆构筑于这两者之上。

4.2 证据组装

探索与撰写之间坐落着证据组装器（fig. 13）。它决定谱系的按节点产物中撰写器可依据者，并将其重塑为单一的面向科学的记录。该设计使这道接缝可审计——某一主张可追溯到哪个节点，是已记录元数据的函数，而非语言模型的判断——从而使 section 4.3 的契约与 section 4.8 的护栏有具体之物可强制执行。

选别按角色进行：一个节点可以不同方式贡献给三个消费者，由单一谓词族判定各自。节点在携带真实测量（或评估器认定其运行成功）时进入 *合成集合*；在相对于父节点新增或修改了源文件时进入 *代码集合*；在其步骤成功时进入 *叙事集合*。仅探索了死胡同的节点不进入任何集合。每条准则都是已记录元数据上的纯谓词，故相同输入总是选出相同的贡献集合。

源文件的收集方式是：沿最佳节点 $n^\star$ 的祖先链 $\mathrm{lin}(n^\star)$，按深度顺序叠覆每个贡献节点触及的文件。由于子节点在父节点工作树的副本内执行，一条相对路径可能出现于多个深度；最深的出现胜出，链上的删除则移除该文件——于是重建集合与最佳节点的实际工作树一致。该选择以相同字节供给两个消费者：撰写器的谱系证据与已发布产物束，因此论文据其描述的谱系代码即为所发布的代码。选择为链上专属——最佳谱系之外的节点即便其测量被合成集合报告也不发布，并记入 provenance——此外为伪代码忠实性，撰写器还另获最高分节点的原始源码，但其本身并非发布集合。

组装器直接从每个节点的工作树读取代码、参数与测量值，而非读取摘要。当实验声明了带类型的结果时，记录将输入参数（问题规模、线程数、随机种子

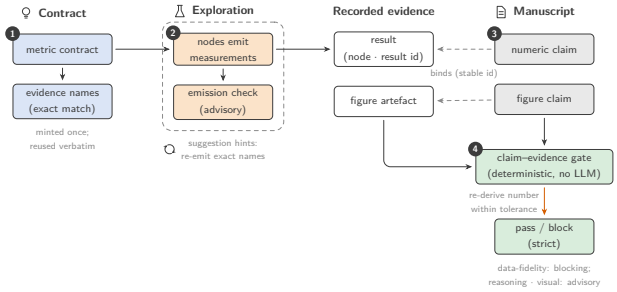

图 14: ARI 对 Story2Proposal 主张–证据契约 [8] 的执行奠基（execution-grounded）实现。(1) 运行级指标契约声明可证伪主张，以及可充当其证据的确切测量名；携带主张的契约一经持久化即逐字复用。(2) 探索节点以这些名称发出测量；一个仅供参考的发出点检查标记仍未覆盖的主张，并提供仅作建议的名称提示以供重新发出——不做任何自动绑定。(3) 稿件主张——数值断言与图引用——通过稳定的节点、结果与图标识符，被绑定到已记录证据。(4) 确定性的主张–证据闸门（无语言模型）在容差内从该证据重新推导每个报告数值；数据保真轴在严格模式下阻断，而推理与视觉一致性轴仅具建议性。

等配置旋钮）与测量输出（吞吐、精度、延迟）及派生量（效率、模型预测上限）分离。此分离并非装饰：跨构型报告的按键极值被确定性地、且仅在测量键上归约，

$$\mathrm{best}(m) \;=\; \operatorname*{red}_{c \in \mathcal{E}} c[m], \qquad m \notin \mathcal{I}, \tag{7}$$

其中 red 按该指标声明的方向取最大或最小，$\mathcal{E}$ 为贡献构型的集合，$\mathcal{I}$ 为从一切归约中剔除的已声明输入键集合。剔除输入正是防止把巨大的输入规模——比如数百万非零元的问题规模——误读为标题结果。语言模型被限定于真正需要它的唯一任务：将逐字源码与日志转化为散文与伪代码，并叙述方法论。它从不提供数值。

支撑这一切的是节点完成时写出的按节点结构化自评报告（检查点内的 `node_report.json`）。它以内容哈希记录：该节点相对父节点新增或修改了哪些文件；节点自身的成功自评连同评估器标记的关切；逐字的构建与运行命令；以及测量所运行的硬件。哈希使源选择可复现，并使已发布产物束能携带由所记录命令构建的复现脚本；所捕获的硬件让论文的环境章节能够基于事实撰写，而非留作“未记录”。

4.3 主张–证据契约

流水线的核心是指标契约：一份运行级声明，列出最终论文必须支撑的可证伪主张，每条主张点名可充当其证据的确切测量名（fig. 14）。这些主张归选定的构思所有，因此实现智能体无法丢弃不便的某条；契约在构思期被持久化到检查点，并贯穿整次运行。探

索期间, 节点以契约的名称发出测量; 同一批名称引导扩展朝向仍未覆盖的主张, 并为 section 3.8 的谱系链接奠基, 使得从较早测量计算得到的证据可以沿单一谱系产生。撰写期间, 下文描述的确定性闸门按确切名称将主张匹配到证据。正面或负面结果都能满足一条主张; 契约强制要求的是用以判定它的证据。

该设计源自 Story2Proposal [8]: 它通过让一份持久契约贯穿由架构师、撰写器、改稿器与渲染器智能体组成的固定流水线, 把一个已完成的「研究故事」——叙事连同其实验证据——转化为稿件。其共享契约把稿件的视觉产物登记到规范标签下, 将其绑定到节级义务, 并携带文档级的验证规则; 面向推理核验、数据保真与视觉一致性的评估智能体观察中间产物, 并在生成过程中更新契约, 而渲染器以确定性的结构验证收尾。该契约所保证的是结构与视觉完整性——唯一标签、可解析的交叉引用、叙事–视觉对齐——其作者对边界亦直言不讳: 该框架「强制结构与视觉义务, 而非生成缺失的科学证据」, 且一篇稿件可以「在结构上有效却包含不够严谨的推理」[8]。

ARI 保留契约思想与三条评估轴, 但把契约重新奠基于执行。所声明的对象不是稿件的视觉结构; 而是运行的证据性义务, 在任何实验运行之前声明, 并绑定到已执行实验实际发出的测量。数据保真随之被重新实现: 不再用语言模型评估智能体, 而是用一道确定性闸门从已记录证据重新推导每个报告数值, 此轴在严格模式下阻断, 而推理与视觉一致性仍仅具建议性。Story2Proposal 验证给定证据如何被写成文;ARI 还控制证据如何形成, 正是这一点使阻断性检查得以成立。

具体而言, 撰写器在每个报告数值旁发出一个文内锚点: 一行注释式声明, 点名该数值支撑的主张、指标、推导公式, 以及它据以计算的操作数构型。确定性的链接器将锚点与已组装的科学记录相调和, 随后闸门用所命名的公式从已记录测量重新计算每个所声明数值, 并在所声明容差内 (默认为一个小的相对容差) 加以比对。该声明是前向唯一的——不做任何反向搜索去猜测某个数值可能意指什么——故错误声明会以不一致失败, 而非被悄然修复。发出也在生产点受检: 当节点发出结果时, 最终闸门日后将运行的存在性检查被立即镜像, 而此时节点仍可重新发出; 警告点名仍未覆盖的主张, 并可携带保守的、仅作建议的词法提示指向所需名称。不做任何自动绑定, 闸门自身也从不查询提示逻辑。

真实运行的经验把这道接缝在四点上加固, 每点都是结构性规则而非提示词修补:

- **契约只铸造一次**。携带主张的契约一经持久化, 后续每次调用都返回逐字的已持久化契约, 而非重新生成; 一次重新推导请求于是变成对义务的无害重读。语言模型的命名不具备指称稳定性——同一个量问两次会得到不同的名字——一次真实运行中, 运行中途的重新生成曾悄然改换证据词汇, 使兄弟节点已发出的测量在精确匹配的闸门面前隐形: 被报告为缺失证据的主张其实早已在更早的名称下被测量过。

- **解析迁就撰写器的实际写法**。闸门的数值核验把科学计数法的两种常见形式 (指数后缀与 $m\times10^k$ 字面量) 解析为单一值, 并把数学定界符与间距宏归一化掉, 使得在提取数值–单位对时数值与其单位相邻。实践中反复出现的撰写器声明癖性——同一推导的公式同义名、带冗余标签前缀书写的操作数引用、携带 LaTeX 转义的指标名、被多处提及共享的单一锚点标识符 (逐次提及消歧)——在解析时被归一化, 而非上报为不一致。其理据有二: 闸门的裁决应度量论文的数字, 而非其表面句法; 且由于锚点行在改稿期不可编辑, 一个在解析时走入死胡同的声明日后绝无法被修复。

- **评审发现完整抵达改稿器**。语义评审——一个互补的语言模型通道, 标记过度主张与缺乏支撑的因果措辞——提出的每条警告都作为一条仅供参考的修订条目被转发给改稿步骤 (section 4.5), 而非截断的子集; 没有被转发条目的警告将永远抵达不了改稿器, 其计数也永远无法下降。改稿后的「已消解」计数是两轮评审之间的原始差值, 因此引入新发现的改稿会显示为负差值——一次回归——而不是被钳制掉。

- **阻断只保留给客观谬误**。闸门发现分两级。受策略约束的发现——未通过重新计算的报告数值、无法解析的操作数、所引证据缺席的数值断言——在严格策略下阻断最终检查, 否则仅报告而不阻断。客观谬误——违反的通用或所声明不变量、失败或缺失的所声明正确性检查、本应是实测上限处却放了占位常量、无法从自身原始操作数重新计算的指标、运行中任何地方都无支撑测量的所声明主张——在 *任何策略下都阻断最终检查*, 因为每一项都是确定性地为假或不成立。语义评审的主观性发现按设计仅具建议性, 引导改稿器, 但从不让发表在判断性问题上被闸门拦下。

4.4 评分细则形式

我们将*评分细则* R 建模为有限树, 叶节点携带元组 $(c_i, w_i, \text{judge}_i)$。每个叶 i 含类别描述符 c_i、非负

权重 w_i 满足 $\sum_i w_i = 1$, 以及返回分级得分的判官函数 $\text{judge}_i : P \to [0,1]$。论文得分即凸组合

$$\text{score}(P) \;=\; \sum_i w_i \,\text{judge}_i(P). \tag{8}$$

此树结构评分细则与 PaperBench [9] 共享——其中二元叶评分以加权平均沿树向上传播;ARI 通过 vendor 化的副本复用上游包 (section 4.6), 使其评分与聚合语义逐字追随上游。判官有两族并存: 面向定性叶的 LLM 评分器, 以及面向具二元或数值校验叶的确定性评分器 (如「成本是否以美元绝对值报告?」)。两者满足同一评估器接口, 因此搜索引擎 (section 3) 与论文流水线通过一个接缝将轴归约为标量。

评分细则以两种模式之一生成。*agent-benchmark* 模式按论文的科学贡献分解树, 各叶询问候选 submission 的执行输出是否再现论文——即原始 PaperBench 范式。*paper-audit* 模式中树的顶层子节点是固定的再现性轴集合, 各叶评分论文文本自身的描述完整性——面向 HPC 再现性审计的范式倒置, 问题不再是「智能体能否再现这篇论文?」而是「这篇论文是否描述了足够再现所需的信息?」。venue 知识 (用哪些轴、如何措辞) 由逐 venue 模板提供而非焊入引擎, 故 ARI 核心保持 domain-agnostic (P4), 新增 venue 是数据变更而非代码变更。

将 paper-audit 得分从退化区域——空 submission 把所有以 submission 为主语的叶推向「否」——提升出来, 需要把叶的问题改为以论文而非 (缺失的)submission 为主语提问、把论文的图与正文一并喂给判官 (使具视觉能力的模型可加以利用)、并允许将可选的 Artifact Description / Artifact Evaluation 附录作为额外输入。这些调整完全留在 ARI 的包装层; vendor 化的判官无改动, 从而使跨上游版本的得分保持可比。我们有意不从论文自身报告值合成再现日志: 那会短路 executed-submission 阶段, 以与 leaderboard 运行不可比的方式抬高得分。

4.5 评审/改稿循环

改稿循环按评审反馈 $J(P_t)$ 迭代地将 P_t 重写为 P_{t+1}:

$$P_{t+1} \;=\; \text{Refine}\big(P_t, J(P_t)\big). \tag{9}$$

当评审者接受草稿——建模为收敛标准 $\text{score}(P_{t+1}) - \text{score}(P_t) < \epsilon$——或达到逐节的小幅改稿上限时, 循环终止。哪条分支更常触发是一个经验问题, section 5 意在以面板规模回答; 此处不做定量主张。

反馈 $J(P_t)$ 是一条合并的修订流, 其来源保持各自分明: 独立的 venue 评审 (section 4.4 的评分细则实例)、被整体转发的证据奠基语义评审 (section 4.3), 以及由闸门发现派生的机械式修订条目。该合并是结构性的——没有语言模型去综合各流——故改稿器以原始术语看到每条发现, 而 venue 评审的独立性也与证据奠基的检查并存。

改稿步骤在每条轴上有意非单调: 修复清晰度问题可能轻微降低某数值轴。这正是 fig. 12 中反向边的来源, 也是收敛标准定义在聚合得分而非逐轴的原因——并配以逐轴下限 (section 4.8), 以防聚合值掩盖单一轴的崩溃。

4.6 PaperBench 集成

复现路径不是对 PaperBench [9] 的重新实现: 上游包被无改动 vendor 化, 使 ARI 逐字追随其任务与评分细则语义, ARI 仅在其周围贡献一层薄包装。包装将协议忠实的循环公开为三个共享词汇的可组合阶段 (fig. 15):

1. *rollout* ——复现智能体从论文出发走向 submission (代码, 以及可选的再现脚本);
2. *reproduce* ——在隔离容器中、于本地机器上或派发到集群执行 submission;
3. *grade* ——vendor 化的判官按评分细则树为 submission 评分。

两个设计选择使其忠于基准。其一, 评分交由 vendor 化判官;ARI 的适配器只整形其输出并把重复运行聚合为单一得分, 故升级 PaperBench 是不改变叶评分结合方式的 vendor swap——同一包络流入 eq. (8), 二元叶取 $\text{judge}_i \in \{0,1\}$。其二, 包装 *fail loud*: 当无法提供基准公平的运行时 (无容器运行时、无调度器、无 GPU 资源登记), 抛出异常而非静默降级到更弱的宿主, 因为静默降级会使所得分数与 leaderboard 条目不可比。旧的 silent-fallback 行为仅经显式 opt-in 可用。

第二个关注点是让复现智能体对宿主机实际提供的能力保持诚实。若放任上游提示词, 智能体倾向于 scaffold 出调用宿主机不存在的二进制的再现脚本, 并不论论文真实语言而默认使用 Python。ARI 通过诱导智能体在 scaffold 前先 probe 环境、并注入由可用工具链/GPU/网络可达性的实时 probe 生成的逐宿主机注记块来打消这两者——使智能体读到的与宿主机实际能做的相一致。

4.7 领域广度: 执行配置契约

PaperBench 上游假定单 GPU 的单容器。为覆盖方法论本质上并行的论文——多 rank MPI 内核、

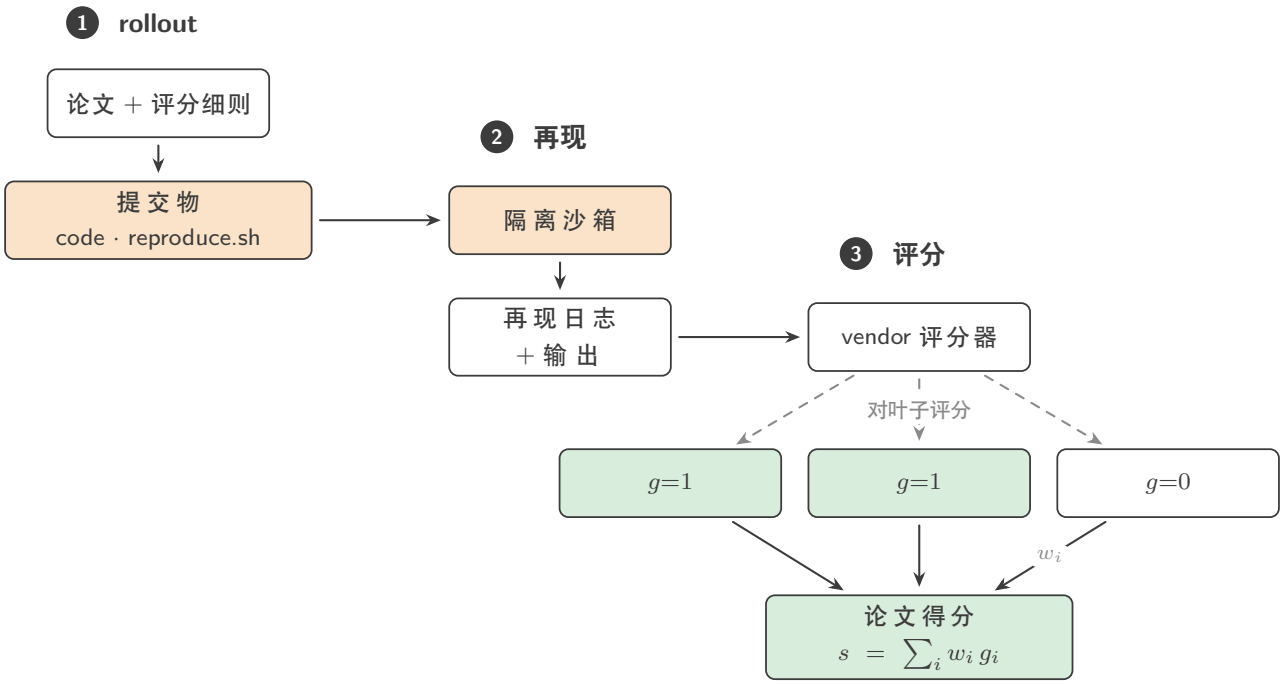

图 15: 将 PaperBench 复现路径表示为三个可组合阶段。智能体的 *rollout* 把论文及其评分细则转化为 submission(代码与再现脚本);*reproduce* 在隔离沙箱中执行该 submission, 捕获再现日志及其输出;*grade* 按加权评分细则树为 submission 评分, 其中 vendor 评分器为每个叶子给出评分 $g \in \{0, 1\}$, 论文得分是按权重聚合的汇总 $s = \sum_i w_i g_i$(eq. (8))。

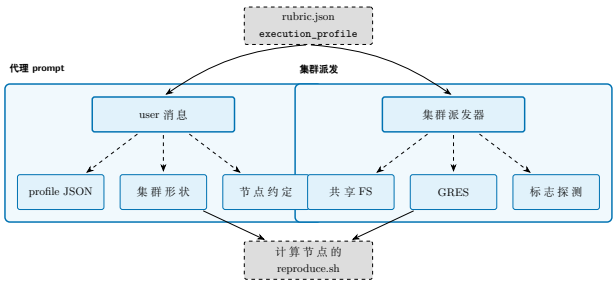

图 16: 执行配置契约从单一评分细则产物流入两个消费方: 复现智能体的提示词 (左) 与集群调度器 (右)。运行时 probe 使派发的资源请求适配本地调度器, 因此一份评分细则在异构集群上无需修改即可忠实运行。

多节点训练、集群特定的模块环境——ARI 以可选的 执行配置 (fig. 16) 扩展评分细则。该配置 domain-agnostic: 它命名的是并行执行形态 (单 CPU、单/多 GPU、MPI、MPI+GPU) 连同论文的规模包络与任何调度器提示, 而非论文科学领域的任何信息。当论文为单机时配置省略, 流水线退化为原始单节点调用: 契约是加法的, 而非破坏性的。

配置供给两个消费方 (fig. 16)。它作为简洁的「计算节点执行约定」块——共享文件系统纪律、优先用调度器的启动器而非裸 MPI 启动器、环境激活、wall-clock 包裹——附加到智能体提示词, 并参数化集群派发 (节点、GPU、内存与放置请求)。由于各集群对接受哪些调度器标志意见不一, 调度器在运行时 probe 本地调度器并丢弃其不支持的标志, 如此同一份评分细则在多 GPU 集群、无 GPU 登记的沙箱分区、或单一工作站上都能忠实派发。一道回归守卫强制其逆命题: 运行于无关集群分配内的 ARI 绝不可把集群或 MPI 特定的文字泄漏进非 HPC 论文的提示词。仅此包装层变化;vendor 化的基准与 ARI 核心在该扩展中保持不变。

4.8 故障模式与护栏

四种故障模式频繁到值得命名。无一是假想的: 最接近的已发表前作记录了所生成草稿误报实验硬件、并在提示词层面已指示仅用真实结果的情况下仍凭空捏造整张结果表 [2]。ARI 的立场是, 每种故障模式都需要结构性——而非仅建议性——的缓解:

- **仅 LLM 支撑**: 论文断言一项谱系中无任何节点佐证的主张。缓解策略将撰写器约束到预先选定的谱系产物集合, 使每项数值或因果主张都可追溯到产生它的节点; 草稿无法引用未被生成的证据。对应的约束在记忆一侧: 已验证上下文 (section 3.10) 仅当结果由产物支撑且未在复现尝试中失败时, 才将其许可为定量主张 (eq. (6)), 因此可复现的结果被优先, 而无支撑的反思无法成为标题数字。
- **引用幻觉**: 论文引用不存在的文献。缓解是结构性的——撰写器不得发明引用键, 且每条文献条目在被采纳前都对外部索引校验 (section 6)。

- **改稿漂移**: 聚合得分上升而论文丧失连贯性。缓解是逐轴下限: 任一轴跌破阈值即中止改稿, 使一轴的提升无法掩盖另一轴的崩溃。

- **数值漂移**: 报告的量与其所概括的已记录结果不一致。由于每项数值主张都点名其所依据的节点与测量值 (section 4.3), 确定性的主张–证据闸门将每项从已记录结果重新推导并在容差内比对 (fig. 14); 与重新计算值不一致的值会被标记, 而 section 4.3 的两级阻断策略决定该发现是在最终检查中拒绝草稿, 还是被报告以待修复。

第一道护栏也限制 context: 撰写器获得选定谱系产物与最高分节点的源码, 且受限于大小预算, 因此即便长时间运行草稿也保持在模型的 context 窗口之内。

5 初步观察

本章报告稼动中的系统在 v0.9.0 所确立的内容。证据分为三类: 一次端到端生成运行, 其发布的论文通过了每一道核验闸门 (section 5.1); 若干由构造直接成立并在运行中得到确认的性质 (section 5.2); 以及一项更早的、在当时的 v0.8.0 流水线上开展的再现练习 (section 5.3)。所有这些都属初步性质: 端到端的观察是单次运行而非受控基准, 它们所呼唤的面板规模研究留待今后工作 (section 5.4)。本报告别处的定量图表仅为示意, 均由固定种子的样本数据生成, 以保证文档构建可复现。

5.1 案例研究: 一篇经闸门核验的示例论文

已有一个端到端结果得到演示并可供查验: 已发布的示例论文——由一次真实运行自主产出的 8 页研究。它考察压缩稀疏行 (Compressed Sparse Row,CSR) 格式的稀疏–稠密矩阵乘法 (SpMM), 在一个由该次运行自身测量拟合而成的性能模型 (论文称之为「loopline」模型) 的引导下, 跨右端项 (RHS) 宽度进行存储策略选型。我们端到端地追踪该次运行 (fig. 17(全文见 section C)), 因为它正是本报告所述核验机制的一个存在性证明。

主张结构。 该次运行的指标契约——在构思阶段一次性铸定, 并在该次运行余下过程中逐字返回 (section 4.3)——声明了 12 条可证伪主张。其中 9 条是单个节点直接测量的独立探针: 对照稠密参考实现的正确性、存储策略对比、一个不规则收集探针与一个局部性破坏探针、一个页面足迹 (转译后备缓冲区,Translation Lookaside Buffer,TLB) 压力探针、一个计算速率探针, 以及若干带宽微基准。余下的 3 条构成一条计算型证据链——拟合性能模型的参数、在留出的探针测量上对拟合作交叉验证, 再从拟合后的模型中选出一个配置——其中每条主张消费前一条的输出, 而非测量某种新的东西。

报告为已落地	正确性: 对照一个独立的稠密参考实现,FP32 最大绝对误差 6.79×10^{-6}; 全部 12 条声明的主张都携带已发出的证据; 计算型证据链已执行。
显式对冲	存储策略的交叉点取值、双精度误差、实测带宽上限, 以及性能模型的数值预测——在本修订中陈述为方法学并标注为未由产物落地。

表 1: 已发布的示例论文只断言经闸门落地的数字, 并在自身正文中对冲其余部分——确定性闸门与评审–改稿循环将论文的范围收敛到该次运行能够支撑的限度。

谱系闭合。 该证据链正是这次运行存在所要演示的情形。一条计算型主张无法由一个没有任何输入的节点来佐证, 而更早的三次运行确认了这一失效模式: 被指示要从同辈探针拟合模型的节点, 转而重跑了单个探针, 链上的主张始终未被覆盖。在此处, 该链得以闭合, 因为一个子节点继承了其父探针节点的工作目录, 并从继承来的测量中计算出拟合、交叉验证与选型——谱系串联 (section 3.8) 使「从你已持有的数据进行计算」成为局部可行的着手方式。随后的最终一轮主张–证据闸门发现全部 12 条主张都已落地并报告零错误, 论文遂得以定稿。

闸门核验了什么, 以及论文作了哪些对冲。 该结果的价值在于这一已核验的性质, 而不在于任何标题数字: 发布论文所断言的每条定量主张都绑定到一份记录产物, 并由确定性闸门重新推导; 凡某项陈述会越出其产物支撑之处, 论文都作了显式对冲 (table 1)。论文将其强定量主张限定于正确性指标——对照一个独立的稠密参考实现, 单精度浮点 (FP32) 的最大绝对误差为 6.79×10^{-6}——并将存储策略的交叉点、带宽上限以及模型的数值预测呈现为方法学, 而非已落地的结果。这种诚实的限定范围正是闸门的效果, 并由语义评审强化: 该评审在草稿中标出 5 处过度主张, 改稿一轮将这 5 处全部消解, 且消解计数被记录为一个原始的前后差值 (section 4.3), 从而一次引入了新过度主张的改稿会作为回归而显现。

我们对这一结果作收窄解读。该性质值得言明, 因

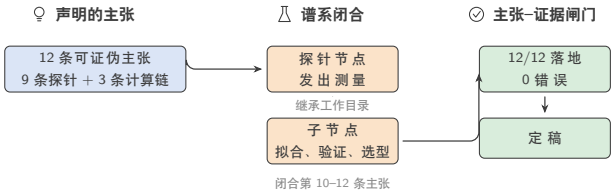

图 17: 案例研究运行，作为主张–证据契约的一个工作示例。12 条声明的主张分为 9 条独立探针与一条 3 条主张的计算链；该链由谱系闭合——一个子节点继承其父探针节点的工作目录，并从继承来的测量中计算拟合、交叉验证与选型——此后确定性闸门发现每条主张都已落地，论文遂得以定稿。

为早先的自主流水线缺少它：在最接近的已发表前驱中，幻觉控制停留在提示层面，对成稿草稿的自动评审只是建议性的而非阻断性的，所生成的论文被观察到会杜撰实验细节，且作者将对所报告结果的自动核验列为今后工作 [2]。单次经核验的运行演示了契约、谱系串联、闸门与评审–改稿循环能够端到端地组合；它并不确立一个分布——那正是下文推迟的受控研究。

5.2 由构造成立的性质

另有三项性质由构造直接成立并在运行中得到确认。成本核算是精确的：每次模型调用都被附加一项美元成本，并按运行汇总进检查点的结果记录。探索终止并非仅凭预算：停滞检测器 (section 3.5) 提供了一种基于内容的退出途径，并已被观察到在步数与成本预算 ($T_{\max}$ / $C_{\max}$) 耗尽之前触发。且执行配置路径按规约行事 (section 4.7)：对一篇高性能计算 (HPC) 论文，评分细则生成会发出一份携带论文所报告规模包络的、多 rank 的消息传递接口 (Message Passing Interface,MPI) 执行配置，而对一篇单机论文则将其省略；同时，一次多标志的集群调度，在运行时 probe 剥离该调度器不支持的标志之后，被本地调度器接受。

5.3 一项更早的再现练习 (v0.8.0)

本系统首次端到端的运行实践早于 v0.9.0：当时的 v0.8.0 流水线经 section 4.6 的再现路径，针对一个已发表的非可逆压缩对象——一个 GPU 上的误差界限受控科学数据压缩器——得到执行。此单一对象上达成的再现程度虽属部分但相当扎实。智能体获取了论文所引用仿真数据集的真实切片，而非以合成数据替代；编译了原生 CUDA 插值内核并在 GPU 上运行；实现了带逐层自动调优的 Lorenzo 与多级插值预测器；并扫掠了一段误差界范围，以相应的压缩比恢复出大约 50 至 90 dB 峰值信噪比 (PSNR) 的重建质量。对照细粒度的评分细则，该次运行仅通过了约十分之一量级的加权叶节点，其原因有两点，皆根植于智能体的行为而非流水线本身：被评分的指标来自一个 CPU 预测器——GPU 内核确已构建并执行，但智能体将其预测质量从所报告的结果中省略——且智能体在仅消耗其时间预算的一小部分后便自行终止，将其余数据集与专有的无损阶段留待未试。这两种行为都吻合复现智能体普遍存在的失效模式：多数模型会过早结束运行并宣称完成，在编写代码上的得分远高于在执行代码并匹配论文结果上的得分 [9]。该练习早于 section 5.1 所述的核验机制，此处原样报告，作为历史基线。

5.4 局限

本评估仍属初步。上述两项结果都不是受控基准：生成运行是单条轨迹，再现练习覆盖单篇目标论文。一项受控评估——在冻结检查点上的中位运行成本、按种子的探索曲线、按类别的评分细则分布，以及在一组对照性目标论文面板上的完整 rollout 复现得分——留待今后工作。同一项研究还应厘清本报告留待解决的经验问题，例如在实践中哪一种终止原因 (section 3.5) 最为频繁地触发。

6 相关工作

我们将 ARI 与四条先行研究脉络对照定位。

6.1 面向代码的树搜索

AIDE [5] 将机器学习工程形式化为代码优化——一个解就是一段由无状态目标函数评分的脚本——并把试错过程铸成对一棵脚本树的搜索，树的边由起草、调试与改进相连。在 MLE-bench [6] 上 (75 项离线 Kaggle 竞赛，按真实奖牌门槛评分)，这一脚手架占据主导：同一模型在 AIDE 下于 8.7% 的竞赛中赢得奖牌，而在两种通用脚手架下分别仅为 0.8% 与 4.4%，可见脚手架设计的分量胜过模型本身的原始能力。ARI 继承了解树的骨架，但将 AIDE 那条被其作者本人标记为局部最优风险的硬编码贪心策略，替换为一个 LLM 主导的选择器 (section 3.3)——它联合权衡 `has_real_data`、整套 `metrics`、深度以及一个软性的 `diversity_bonus`——并加上 section 3.7 的跨运行谱系。而当两份工作都随任务一并领受其目标时——MLE-bench 的作者也承认，真实研究“甚至可能并没有一个清晰的问题陈述”——ARI 自行铸造目标：指标契约 (section 4.3) 在搜索开始之前就声明了何者方能算作证据。

AlphaCode [10] 是生成-过滤范式的经典成果: 每道题采样多达数百万段程序, 在题目给出的示例测试上过滤, 并聚类到至多十份提交, 在十场模拟 Codeforces 竞赛中达到平均排名位于参赛者前 54.3%。该配方预设了一个廉价的、由任务提供的验证器; 而一个 ARI 候选乃是在集群资源上执行的一次实验, 并无现成的验证器, 因此广度让位于对一个小前沿的最佳优先引导, 廉价过滤也让位于针对该运行自身证据的确定性检查。

6.2 自主研究智能体

AI Scientist 系列 [2, 3] 端到端地闭合了一个类似的循环 (idea → experiment → paper), 据报告每篇论文的成本低于若干美元, 并配有一个在真实会议投稿上达到近乎人类平衡准确率的自动评审。但该评审是建议性的——它对成品 PDF 排序, 而幻觉控制停留在提示词层面——其作者在罗列了被臆造的硬件、被捏造的消融表, 以及一个被叙述为改进的负面结果之后, 将结果的自动验证列为未来工作。ARI 是其逆否命题: 实验在 BFTS 下分叉, 而非每个想法只有一条线性轨迹; 指标是一份在首次铸造时即冻结的运行级工件 (section 4.3); 且验证会阻断——决定何者可被发表的, 是一道确定性的 claim–evidence 闸门, 而非一个评审分数 (section 4.8)。

Agent Laboratory [4] 将智能体视为协作者而非驱动者: 它要求人类提供一个研究想法, 在一个 LLM 奖励模型 (对计划遵循度评分) 之下变异一个扁平的双程序池, 并以模拟评审来把守论文。它最具启发性的结果是负面的: 那些评审在十分制上比人类评审高估了 2.3 分——这是对何以整体式 LLM-as-judge 把守并不充分的最清晰量化。因此 ARI 的语义评审只引导而绝不阻断, 阻断被保留给确定性闸门; 并且每个节点都持久存于一棵检查点作用域的树中, 而非一个瞬时的池里。

VirSci [7] 覆盖了这些系统处理得最为单薄的阶段: 一支以某研究社群的数字孪生为根基的科学家团队, 其多智能体研讨在与当代研究的契合度和潜在影响力上, 较单智能体基线分别提升了 13.8% 与 44.1%。它止步于一份由基于嵌入邻近度的新颖性代理指标评分的摘要, 其作者也呼吁更严格的下游验证。ARI 取下了这一分工的两半: 它直接且不加修改地驱动 VirSci 原始的研讨引擎 (fig. 18)——基于新鲜度的 `select_coauthors` 团队组建, 以及多智能体的 `generate_idea` 循环——运行于一份实时的 Semantic Scholar 快照之上, 从而其所依托的机制乃是已发表者本身在当前文献上的执行; 同时它在下游补上那

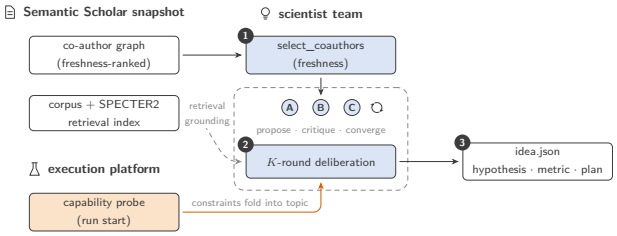

图 18: ARI 所驱动的 VirSci 式想法生成。`select_coauthors` 从一份冻结的 Semantic Scholar 快照 (带 SPECTER2 检索索引的语料, 以及按新鲜度排序的合著者图) 组建一支异质的科学家团队; 团队进行以对该快照的检索为奠基的 K 轮研讨, 而一次运行开始时的能力探测把平台被测得的约束折入研讨主题, 使得收敛后的 `idea.json` 只承诺平台能够进行的测量。

被推迟的验证, 在那里,claim–evidence 闸门下的已执行证据取代了代理式的新颖性评分。

6.3 论文评估

PaperBench [9] 开创了评分细则即树的形式: 其 20 篇目标论文各自携带一份与作者共同开发的评分细则——共计 8,316 个可评分的叶子节点——其叶级评分为通过/不通过, 并按加权平均聚合, 且设有一个洁净起始的复现阶段, 以使硬编码的结果无法冒充复制。ARI 原样引入该基准 (section 4.6), 采用相同的按叶分级与聚合权重 (eq. (8)), 并加上一个用作改稿护栏的按轴下限 (section 4.8)。它揭示的执行鸿沟——最强的智能体在代码开发上得分 43.3%, 但在执行上仅 4.5%, 在匹配结果上为零——正促使将验证移入生成之内, 而非事后评分; 且其手工搭建的评分细则 (每篇论文耗时数周, 并需原作者参与) 将自动化评分细则构建留作未来工作, 而这正是 ARI 所取的分支: 评分细则按论文逐一生成 (section 4.4), 因此手工评分细则的复现与可与排行榜相比较的分数, 在设计上即不在范围之内。

Story2Proposal [8] 通过一份持久的共享契约——章节结构、视觉工件的注册表、全文档级的校验规则——来统辖结构化的稿件生成, 由 writer、refiner 与 renderer 智能体在一个以推理验证、数据保真度与视觉一致性评分的 generate–evaluate–adapt 循环下协同。其增益在四种 LLM 主干上均成立, 但证据被作为给定输入接收, 而其失效模式分析也承认一份稿件可以 "在结构上有效, 同时所含推理却不够严谨": 该契约约束的是结构, 而非证据上的有效性。ARI 将这一原则及其各评估轴带入一个以执行为基础 (execution-grounded) 的流水线 (section 4.2): 契约为实验记录补充引用稳定的节点、结果与图表标识符的 claim 与 nu-

meric assertion; 数据保真度成为从已记录结果重新推导每个报告数值的确定性闸门; 推理与视觉一致性成为独立稿件评审之外一道不阻断的、以证据为根基的评审。由于 ARI 亲自执行实验, 它把这份稿件级契约扩展到了以执行为基础的溯源 (provenance)。

6.4 基础

每节点的智能体循环 (section 3.6) 遵循推理-行动 (Reason-and-Act,ReAct) 模式 [1], 其确切发现是一种权衡: 将思考与有根据的行动交替进行, 在显著削减幻觉的同时抬高了推理错误率。改稿循环承自 Self-Refine [11] 与 Reflexion [12], 二者各带一条 ARI 认真对待的告诫:Self-Refine 的增益在模型无法察觉自身错误之处便会消失, 而 Reflexion 的消融实验表明, 驱动其改进的是有根据的评估——而非记忆本身。因此 ARI 绝不让模型在事实问题上为自己评分: 改稿由评分细则分数与闸门裁决引导, 二者均在撰写者之外计算, 而一项结果唯有在以工件为根基时才成为定量 claim (section 3.10)。思考步中的草稿板承自思维链 (chain-of-thought) 提示 [13], 其增益被报告为随模型规模而涌现——其作者从未声称模型在进行推理。这些工作无一提出超出单条轨迹的状态; 那使一次运行可复现的检查点作用域状态, 是 ARI 的贡献。

早期草稿曾另引一份研究智能体基准, 但当没有候选来源能通过 section 4.8 的参考文献校验规则时, 该引用被移除, 因此此处的覆盖被限定为可经 Semantic Scholar 索引校验的条目。

7 讨论与局限

7.1 确定性 vs 新颖性

P2, 即确定性原则, 是系统最具约束力的约束。它约束的并不是研究记忆 (section 3.10) 是否可以使用嵌入模型, 而是由此产生的非确定性可以在哪里起作用: 记忆子系统不做任何生成式语言模型调用, 而其按嵌入相似度排序的归档检索——因依赖远端服务, 故并非按字节可复现——被排除在可复现路径之外, 仅作为一项可选的回忆辅助, 循环从不取用它, 其本身也无法为论文主张提供依据。循环真正依赖的那些记忆读取——在每个节点注入的工作上下文, 以及为论文提供依据的已验证上下文——是节点级报告的确定性、祖先作用域折叠。每一项可复现性的胜利都关闭了这样一道通往非确定性的门; 我们接受这一代价, 因为契约对象是检查点, 而非运行。

7.2 结构接缝处的指令遵循

v0.9.0 反复印证的教训是: 指令遵循在结构接缝处会退化——需要解析的一个索引、需要复用的一个名称、需要声明的一个数字——而持久的补救之道是在解析端吸收, 而非依靠提示词。一个无法解析的选择索引会回退到一个确定性排序 (section 3.3); 一个模型被问两次会以不同方式命名同一个量, 因此指标契约只铸造一次, 而后逐字重放; 写作器在数值声明上反复出现的怪癖会在解析时被规整 (section 4.3)。所吸收怪癖的目录是被动的, 从真实运行中逐步积累, 因此一个未曾预料的变体首先会显现为一处关卡不匹配——这是保守的做法: 它会被标记, 一处未解决的不匹配会阻断最终检查, 任何东西都不会被悄无声息地接受。

同一套纪律也固定了什么可以构成阻断。关卡证明每一项定量主张都与一份记录在案的产物相符, 而语义评审的主观发现引导精炼器 (section 4.5) 但从不对发表构成关卡——这一点不同于 [2] 那种完全事后的评审——因为在阻断路径上的主观裁决会是一道无法审计的否决权。因此, 一篇经过验证的论文仍然可能是一篇乏味的论文。

7.3 扩展极限

支撑 BFTS 运行循环的单机假设是最大的扩展约束: 单个 `ThreadPoolExecutor`(section 3.2) 是唯一的工作池, 上限为四个工作线程。多机扩展需要将显式谱系纪律 (P4;section 2.5) 推广到合并, 而不仅是祖先遍历——这同一道缺口也限制了计算得出的证据, 其证据链 (section 3.8) 如今是沿单一谱系按名称提示调度的, 而非由一个在证据依赖图之上的显式调度器来调度。另一项约束是 LLM 评分器, 经验上它与探索器并列为占主导的开销项之一; 按 (论文哈希, 细则哈希) 缓存其输出会有所帮助, 但此特性尚未发布。

7.4 展望

最大的缺口是评估广度:section 5 建立在一次复现实验 (在彼时最新的 v0.8.0 流水线上运行) 和一篇经过验证的生成论文之上——这是机制能够组合起来的证据, 而非一个分布。未来工作有三方面: 一项在冻结检查点之上、以一组相互对照的目标论文为面板的受控研究, 其协议精神与 [6, 9] 一脉相承; 以对重复 rollout 与评分器轮次的、具方差意识的聚合, 取代单一标量的汇总, 因为评判者本身就带有噪声; 以及更广的领域——把契约可命名的证据种类从标量测量进一步拓宽, 正如执行画像契约 (section 4.7) 拓宽了

ARI 所能承载的计算那样, 迈向那些主要证据并非单一指标的领域。

Algorithm 1 BFTS run-loop(单次外层迭代)。

Input: 状态 $S = (\mathcal{F}, \mathcal{P}, e, H)$; 配置 $(B_N, B_D, E_{\max})$ 与 worker 上限 `max_parallel` $= \min(\texttt{max\_parallel\_nodes}, 4)$

Output: 更新后状态 S'

▷ — 内层扩展子循环 —

while $\mathcal{F} \neq \emptyset \wedge |\mathcal{P}| < \texttt{max\_parallel} \wedge |\mathcal{N}| < B_N$ **do**
  $f \leftarrow \text{select}^{\text{LLM}}(\mathcal{F})$ ▷ `select_best_to_expand`
  if $\Pi(f; |\mathcal{N}|)$ **then**
    $\mathcal{F} \leftarrow \mathcal{F} \setminus \{f\}$; **continue**
  end if
  $c \leftarrow \texttt{expand}(f, \texttt{depth\_note}, \texttt{budget\_note})$
  $\mathcal{P} \leftarrow \mathcal{P} \cup \{c\}$; $e(f) \mathrel{+}= 1$
end while

▷ — 外层并行执行批次 —

$B \leftarrow \emptyset$
while $|B| < \min(\texttt{max\_parallel}, |\mathcal{P}|)$ **do**
  $n \leftarrow \text{select}^{\text{LLM}}(\mathcal{P})$ ▷ `select_next_node`: 单个节点
  $\mathcal{P} \leftarrow \mathcal{P} \setminus \{n\}$; $B \leftarrow B \cup \{n\}$
end while
for all $n \in B$ **并行 do**
  $n' \leftarrow \text{ReAct}(n)$ ▷ `AgentLoop.run`, 可能失败
  $H \leftarrow (H \,\|\, \text{label}(n'))[-W:]$ ▷ `record_run`
  $\mathcal{F} \leftarrow \mathcal{F} \cup \{n'\}$
  for all $p \in \mathcal{F} : p = \text{pa}(n')$ **do** ▷ 规则 A
    if $r(n') > r(p)$ **then**
      $\mathcal{F} \leftarrow \mathcal{F} \setminus \{p\}$
    end if
  end for
  for all $p \in \mathcal{F}$ **do** ▷ 规则 B
    if $e(p) \geq E_{\max}$ **then**
      $\mathcal{F} \leftarrow \mathcal{F} \setminus \{p\}$
    end if
  end for
end for
return $S' = (\mathcal{F}, \mathcal{P}, e, H)$

Algorithm 2 证据组装 (探索 → 撰写器输入)。

Input: 带按节点报告的谱系节点 $\mathcal{N}$; 指标方向映射

Output: 科学记录 $\mathcal{E}$: 带类型构型、按键极值、方法论叙事

$n^\star \leftarrow \arg\max_{n \in \mathcal{N}} r(n)$ ▷ 同分: 已验证, 其次更深

$L \leftarrow \mathrm{lin}(n^\star)$ ▷ 根到最佳的祖先链

$\mathcal{C} \leftarrow \{\, n \in L : \textsc{Contributes}(n) \,\}$ ▷ 角色谓词 (代码)

$F \leftarrow \emptyset$ ▷ 相对路径 $\mapsto$ (节点, 深度)

for all $n \in L$ 按深度顺序 **do**

  for all $n \in \mathcal{C}$ 新增/修改的路径 p **do**

    if $p \notin F \vee \mathrm{depth}(n) \geq \mathrm{depth}(F[p])$ **then**

      $F[p] \leftarrow (n, \mathrm{depth}(n))$ ▷ 最深胜出

    end if

  end for

  for all n 删除的路径 p **do**

    从 F 移除 p ▷ 删除以叠覆移除

  end for

end for

在尺寸预算下逐字读取每个 $p \in F$ 的字节

for all 测量键 $m \notin \mathcal{I}$ **do**

  $\mathrm{best}(m) \leftarrow \mathrm{red}_c\, c[m]$ ▷ eq. (7); 确定性

end for

$\textsc{Narrate}(F)$ ▷ LLM: 逐字源码/日志 → 散文、伪代码

return $\mathcal{E}$

A 记号表

本附录汇集正文中使用的全部记号及其含义。

符号	含义
$\mathcal{N}$	搜索树的节点集
$\mathcal{T}$	搜索树 $(\mathcal{N}, E)$
$n \in \mathcal{N}$	单一节点
$\mathrm{ch}(n)$	n 的子集
$\mathrm{pa}(n)$	n 的父节点
$\mathrm{depth}(n)$	n 的深度
$s(n)$	节点状态 $\in \{\mathrm{open}, \mathrm{eval}, \mathrm{done}, \mathrm{failed}\}$
$r(n)$	n 的标量奖励
$\mathbf{e}(n)$	按轴评估向量
ϕ	轴 $\to$ 标量聚合器
$\mathrm{div}(n)$	多样性奖励 (eq. (4))
$\mathrm{fb}(n)$	确定性回退排序 (eq. (5))
$T_{\max}, C_{\max}$	步数 / 成本预算
R	评分细则树
c_i	第 i 叶的类别
w_i	第 i 叶的权重
judge_i	第 i 叶的判官
P	论文草稿
$\mathrm{score}(P)$	论文聚合得分 (eq. (8))
Refine	改稿算子 (eq. (9))
ckpt	检查点目录
$\mathrm{lin}(n)$	终止于 n 的谱系链

B 运行时 LLM 提示词

本附录列出 ARI 在运行时所用的 LLM 提示词, 逐字摘自 v0.9.0 时刻的 `ari-core/ari/prompts/`。每段清单与磁盘源码逐字节一致。因为输入模型的字节本身决定其行为, 提示词不进行翻译, 在本报告的各语言版本中均以原文原样呈现。

编排器

B.1 BFTS 扩展

```
You are expanding a BFTS research tree node.

{goal_line}Parent node id={parent_id_short}, depth={parent_depth}, status={parent_status}
{depth_note}{budget_note}Parent metrics: {parent_metrics_json}
Parent summary: {parent_summary}
{sci_note}{idea_block}{parent_report_block}
{siblings_block}{ancestors_block}{existing_block}{diversity_block}Propose exactly ONE child research direction
that is the most scientifically valuable next step. For the "label" field, strongly prefer one of the canonical
labels [draft, improve, debug, ablation, validation]; only invent a custom label (e.g. "replication",
"generalization") when none of the five fits meaningfully — the custom string will be preserved as raw_label.
Base your choice on the experimental context above, not on a fixed template.
```

```
Label selection guidance:
- 'debug': parent FAILED or has no real data — diagnose and fix it.
- 'improve': parent succeeded and you want to push its metrics higher by tuning parameters, flags, or
algorithms.
- 'ablation': isolate which component drives the parent's gains by removing or varying ONE component. State
explicitly what is removed/varied and what delta vs. the parent metrics you expect.
- 'validation': rigorously verify the parent's claims (different seeds, edge cases, stress tests,
expected-degradation checks).
- 'draft': start a fresh implementation from scratch to introduce a fundamentally NEW perspective (use this
instead of inventing a new label like 'replication' or 'generalization').

Reply ONLY with a JSON array containing exactly one element: [{{"label": "<one of:
draft|improve|debug|ablation|validation>", "direction": "..."}}]
Example: [{{"label": "validation", "direction": "<one-sentence plan>"}}]
```

B.2 BFTS 扩展期选择

```
You are selecting which completed research node to expand next in a BFTS tree.

Experiment goal: {experiment_goal}

Completed nodes awaiting expansion:
{candidates}

Select the single most promising node to expand. Prefer nodes with high scientific_score, strong metrics, and
unexplored directions. Avoid nodes that have already been retried many times or are at excessive depth.
Reply with ONLY the index number (0-based).
```

B.3 BFTS 选择

```
You are selecting the most promising node to explore next in a research tree.

Experiment goal: {experiment_goal}

Relevant past memories:
{memory_context}

Candidates:
{candidates}

Selection criteria:
- Nodes with has_real_data=True and strong metrics are high-value
- Consider all metrics holistically (multi-objective)
- Deeper nodes with excellent results are worth continuing
- A small diversity_bonus is awarded to underrepresented exploration types; treat it as a soft tiebreaker, not
a primary signal
Reply with ONLY the index number (0-based) of the best node.
```

B.4 谱系续延判断

```
You are a research orchestrator. Given the current state of an exploratory research run, decide the next
lineage-level action. Possible actions:
  continue       — current idea is still productive; keep exploring
  switch_to_idea — current idea has stagnated; switch to an alternative
  fanout         — current idea succeeded; explore an alternative in parallel
  terminate      — research thread is exhausted; stop
```

```
Choose the LEAST disruptive action that fits the evidence. Prefer continue unless there is a concrete reason to
escalate. When choosing switch_to_idea or fanout, set target_idea_index to a value that appears in the
alternatives pool. When choosing switch_to_idea, set disable_generate_ideas=true (the child should run with the
chosen idea pinned, not regenerate). For fanout you may set it false so the child can also explore additional
novel directions.
Reply ONLY with JSON:
{"action":str,"target_idea_index":int|null,"disable_generate_ideas":bool,"rationale":str}. No markdown.
```

B.5 根创意选择

```
You are a research orchestrator picking the ROOT idea for a run from a VirSci-generated pool. VirSci has
already scored each idea by novelty/feasibility/clarity, but you have additional context (venue rubric,
ancestor research thread, run notes). Your job is to pick the idea most likely to produce a strong submission
for this venue, considering all signals.

Rules:
  - chosen_index MUST be an integer that exists in the pool.
  - Default to VirSci's choice (index 0) UNLESS another idea is     clearly stronger given the additional
  context.
  - One sentence rationale describing your reasoning.
Reply ONLY in JSON: {"chosen_index": int, "rationale": str}. No markdown.
```

智能体循环

B.6 ReAct 智能体系统提示

```
You are a research agent. You MUST use tools to execute experiments. Do NOT write plans or text descriptions —
call a tool immediately.

AVAILABLE TOOLS:
{tool_desc}

RULES:
- Your FIRST action must be a tool call. Never output a text plan.
- If `make_metric_spec` tool is available and this is a new experiment (not a continuation), call it early to
self-determine evaluation criteria.
- NEVER fabricate numeric values — only report values from actual tool outputs
- AFTER your final measurement run, when `emit_results` is available, call it once to record a typed split
between INPUT parameters (matrix size, thread count, seeds — knobs you ran on) and MEASUREMENTS (throughput,
accuracy, latency — what you measured). This lets downstream stages distinguish "what we ran on" from "what we
measured" so a best-of reduction never picks an input size as the result. Do NOT include input parameters in
`measurements` and do NOT include measured outputs in `params`.
- When all experiments are done, return JSON: {{"status": "success", "metrics": {{...}}, "summary": "..."}}
- Do NOT call gap_analysis or generate_hypothesis
- Ensure your experiment is reproducible: capture whatever information would be needed for an independent
researcher to reproduce your results and verify your findings{memory_rules}{extra}
```

评估器

B.7 同行评议评估器

```
You are a research data extractor AND a scientific peer reviewer.
Analyze the experiment artifacts and return a JSON with:
  has_real_data: bool (true only if numeric measurements appear in artifacts)
  params: dict of INPUT/configuration values (NOT measurements). Examples: matrix size, thread count, seed.
  measurements: dict of MEASURED quantities (the experiment's output). Examples: GFLOP_per_s, accuracy,
  latency.
```

```
  metrics: dict — flat union of params and measurements (back-compat).
  reason: str (one sentence describing what was measured)
{axes_block}
  comparison_found: bool (true if results involve comparison with existing approaches)

When scoring claim_implementation_alignment, look for concrete preconditions or contracts the plan / model
states (e.g. 'eliminates RFO traffic', 'preserves equivariance', 'requires sorted input') and cross-reference
them against the artifacts. Score low when the implementation contradicts a stated assumption, even if the
kernel runs without errors.
Return ONLY valid JSON, no markdown fences.
```

B.8 指标抽取

```
You are a research data extractor AND a scientific peer reviewer.
Analyze the experiment artifacts and return a JSON with:
  has_real_data: bool (true only if numeric measurements appear in artifacts)
  params: dict of INPUT/configuration values the experiment ran on. These are knobs, not outcomes. Examples:
  matrix dimensions (M, K, nnz), thread count, batch size, ISA flags, seeds. They are NOT candidates for a
  best-of reduction.
  measurements: dict of MEASURED quantities — what a peer reviewer would treat as the experiment's result.
  Examples: GFLOP_per_s, GB_per_s, latency_s, accuracy. ARE candidates for best-of.
  metrics: dict — for back-compat, the union of params and measurements as a single flat dict (e.g.
  {{"GFLOP_per_s": 754.8, "M": 120000}}). Downstream consumers may read either the typed split above or this
  flat view. NEVER include the same key in both views with different values.
  reason: str (one sentence describing what was measured)
  axis_scores: dict with EXACTLY these five keys, each a float in 0.0-1.0:
    - measurement_validity: are numeric measurements present and methodologically sound?
    - comparative_rigor: are results compared against baselines or prior work?
    - novelty: does the work advance beyond existing approaches?
    - reproducibility: is there enough detail for someone else to reproduce the result?
    - clarity_of_contribution: is the scientific claim stated clearly and specifically?
    Score 0.0 on an axis when that dimension is absent or fatally weak.     Score toward 1.0 as the dimension
    approaches publishable quality.     These axes are combined via weighted harmonic mean, so a low score on
    ANY axis     drags the overall score down — differentiate axes deliberately.
  axis_rationales: dict with the same five keys; each value is one sentence     explaining the score for that
  axis.
  comparison_found: bool (true if results involve comparison with existing approaches)
Return ONLY valid JSON, no markdown fences.
```

流水线

B.9 关键词管理员

```
You are a research librarian. Given experiment summaries, produce a SHORT broad academic search query (3-6
words) for Semantic Scholar. Use GENERAL terms (e.g. 'algorithm optimization', not 'custom 4-stage pipelined
batch-norm fused kernel'). Avoid domain-specific jargon, acronyms, or overly narrow terms. Return ONLY the
query string, no explanation.
```

向导 (Viz)

B.10 向导目标对话

```
You are an assistant helping a researcher set up an ARI experiment. ARI automatically writes, runs, benchmarks code, then produces a paper. Ask focused questions ONE AT A TIME to understand: (1) what to optimize or investigate, (2) how to measure success (metric name and direction), (3) platform/hardware constraints, (4) any baseline to compare against. Be concise. After 3-5 exchanges and sufficient info, output EXACTLY: ---READY--- followed by a complete experiment.md with sections: ## Research Goal, ## Evaluation Metric, ## Constraints. Do NOT output the MD before getting at least 2 user responses.
```

B.11 向导配置生成器

```
You are helping a researcher set up an automated experiment. Convert the following research goal into a concise experiment.md file with a ## Research Goal section. Keep it to 3-5 sentences maximum. Do not add code or technical details. Research goal: {goal}
```

C 生成的示例论文

本附录完整、未经编辑地收录系统在 section 5.1 的案例研究运行中自主生成的全 8 页论文。它作为产物被收录——论文所作的每一条论断都通过了确定性的 claim–evidence 闸门, 凡是超出已记录证据的陈述, 正文都显式作了对冲。硬件仅以指令集层面描述, 唯一出现的产品名出现在被引论文的标题之中, 而非对运行环境的描述。

CSR Sparse-Dense Matrix Multiplication on CPUs with RHS-Width Robustness and a Loopline-Guided Performance Model

Autonomous Research Infrastructure

June 12, 2026

Abstract

Sparse-dense matrix multiplication (SpMM) in CSR format is a core kernel in scientific computing and modern sparse learning pipelines, yet its performance on CPUs is highly sensitive to the width k of the dense right-hand side (RHS) matrix. We present a CSR SpMM implementation with a cache-aware kernel structure and a store-policy selector intended to switch between regular temporal stores and non-temporal stores as k varies. We validate correctness against an independent dense FP32 reference, achieving a maximum absolute error of 6.78958×10^{-6}.

1 Introduction

Sparse-dense matrix multiplication (SpMM), $C \leftarrow AB$ with sparse A and dense B, is widely used in simulation codes, signal processing, and learning workloads. In CSR SpMM, the irregular access pattern induced by the sparse structure interacts with cache hierarchies and store buffers; a particularly important practical issue is that performance can change drastically when the RHS width k changes (e.g., small k behaves like SpMV, while larger k exposes more reuse in B). This sensitivity complicates performance portability and makes it difficult to choose optimizations such as non-temporal stores and blocking strategies.

This paper targets a CPU implementation of CSR SpMM with a cache-aware kernel structure and a store-policy selector intended to adapt to varying k, and it outlines a measurement methodology intended to support future performance modeling (e.g., identifying bandwidth- vs. compute-limited regimes). Our work is motivated by the broader importance of sparse matrix kernels Dorrance et al. [2014], Alappat et al. [2020] and by recent interest in high-performance SpMM on modern Arm CPUs Lei et al. [2025].

Contributions.

- We implement a CSR SpMM kernel and a store-policy selector intended to choose between regular temporal stores and non-temporal stores as a function of k (Figure 1 illustrates the design goal).
- We provide a reproducible measurement methodology that includes correctness validation against an independent dense FP32 reference, plus optional microbenchmarks (bandwidth probes and a compute-inflation probe) intended to support future performance modeling.
- We provide an experimental setup for collecting artifacts on aarch64 and x86_64 to support future characterization and model validation Alappat et al. [2020], Dufek et al. [2021].

2 Related Work

Sparse matrix kernels have a long history of architecture-aware optimization, where performance is often limited by memory bandwidth and irregular access rather than peak FLOPs. Foundational studies of sparse kernels emphasize scalable implementations and the role of memory systems Dorrance et al. [2014]. On recent aarch64 HPC systems, performance modeling for streaming kernels and sparse operations highlights the importance of measuring attainable bandwidth ceilings and accounting for architectural details beyond nominal peak values Alappat et al. [2020]. Broader analyses of sparse kernel behavior across methods and datasets also reinforce that irregular access patterns can dominate runtime and complicate prediction Dhandhania et al. [2021].

For sparse matrix-matrix multiplication, modern work explores format choices and heterogeneity-aware strategies Cheng et al. [2023], Namjoo et al. [2026]. While much of this literature focuses on SpGEMM or GPU/heterogeneous settings, the underlying challenge of selecting appropriate kernels and data layouts remains central on CPUs, particularly as k changes. Recent work on Arm architectures argues that careful microarchitectural tuning is required for high SpMM performance Lei et al. [2025]. Our work complements these efforts by focusing on implementing mechanisms intended to improve k-robustness in CSR SpMM (notably a store-policy selector), and by providing a lightweight measurement methodology intended to support future model development and validation.

3 Methodology

We compute $C \in \mathbb{R}^{M \times k}$ from a CSR matrix $A \in \mathbb{R}^{M \times K}$ and a dense RHS $B \in \mathbb{R}^{K \times k}$:

$$C_{i,:} \leftarrow \sum_{p=\texttt{rowptr}[i]}^{\texttt{rowptr}[i+1]-1} \texttt{val}[p] \cdot B_{\texttt{col}[p],:} \tag{1}$$

The implementation is parallelized over CSR rows using OpenMP with a fixed thread count.

3.1 CSR SpMM kernel

The baseline kernel follows the standard CSR SpMM structure: for each row i, iterate over the nonzeros, gather the corresponding RHS row of B, and perform a scaled vector add into $C_{i,:}$. The critical design choice for k-robustness is how the kernel initializes and stores C, because C is dense and potentially large; depending on k, write-allocate and cache pollution can dominate.

We consider two store policies:

- **Regular temporal stores**: standard writes to C, benefiting from cache when reuse exists.
- **Non-temporal stores**: streaming stores to reduce cache pollution and avoid write-allocate traffic for large k.

A selector chooses the policy based on an empirical tuning procedure over k (Section 3.4); in this revision we describe the mechanism, but do not claim artifact-grounded crossover values.

3.2 Correctness validation

Correctness is validated by comparing the CSR SpMM output against an independent dense reference implementation (dense A constructed from CSR) in FP32. FP64 validation is an optional

check when enabled, but FP64 error results are not reported as artifact-grounded outcomes in this revision. The reported metric is the maximum absolute elementwise error:

$$\epsilon_{\max} = \max_{i,j} \left| C_{ij}^{(\text{spmm})} - C_{ij}^{(\text{ref})} \right|.$$

3.3 Microbenchmarks and ceilings

To parameterize the performance model, we measure:

- **DRAM memcpy bandwidth** and related in-code memory ceilings (used as bandwidth ceilings).
- **STREAM triad bandwidth** as an additional indicator of sustainable bandwidth.
- **Compute inflation probe**: an artificial increase in arithmetic intensity by adding a fixed number of fused multiply-add operations (FMAs) per element of B. This yields an estimate of an effective compute rate f_{eff} (FLOP/s) that can be used as a compute ceiling.

This style of combining attainable bandwidth measurements with compute ceilings follows the spirit of roofline/loopline modeling Alappat et al. [2020], Dufek et al. [2021].

3.4 Store-policy selector

For a fixed sparse matrix A and varying k, we benchmark both store policies and identify an empirical crossover k at which non-temporal stores become faster than regular stores. The selector then chooses:

$$\texttt{policy}(k) = \begin{cases} \texttt{regular} & k < k^{\cdot} \\ \texttt{non-temporal} & k \geq k^{\cdot} \end{cases}$$

Figure 1 visualizes the speedup trend and the crossover behavior.

3.5 Reproducible pseudocode

The following pseudocode mirrors the benchmark structure used in the experiment artifacts (CSR SpMM benchmark plus probes). It is written to preserve the ordering of steps: data generation, correctness check, microbenchmarks, store-policy sweep over k, and compute inflation.

Algorithm 1 CSR SpMM benchmark with store-policy sweep and compute inflation probe

Inputs: thread count T; CSR dimensions (M, K); `nnz_per_row` or `nnz`; RHS widths $k \in \{8, 16, 32, 64, 128\}$; warmup w; iterations r; seeds
Set `OMP_NUM_THREADS` $\leftarrow T$
Generate CSR matrix A with fixed `nnz_per_row`: for each row i, sample `nnz_per_row` column indices and values using the given seed; build `rowptr`, `col`, `val`
Generate dense RHS $B \in \mathbb{R}^{K \times k}$ with pseudo-random values (seeded)
Allocate $C \in \mathbb{R}^{M \times k}$
Correctness check:
Build dense reference A_{dense} from CSR (or compute reference directly) and compute $C_{\text{ref}} \leftarrow A_{\text{dense}} B$
Compute $\epsilon_{\max} \leftarrow \max |C - C_{\text{ref}}|$ for FP32; optionally repeat in FP64 when enabled (FP64 error is not reported as a grounded result in this revision)
Bandwidth microbenchmarks:
Measure DRAM `memcpy` bandwidth and (optionally) `memset` bandwidth on arrays sized to exceed cache
Measure STREAM triad bandwidth
Store-policy timing sweep:
for each k in $\{8, 16, 32, 64, 128\}$ **do**
    Regenerate/resize B and C for this k
    Time CSR SpMM with **regular temporal stores**: run w warmup iterations, then average over r iterations
    Time CSR SpMM with **non-temporal stores**: run w warmup iterations, then average over r iterations
end for
Determine empirical crossover k where non-temporal becomes faster; record speedup curve
Compute inflation probe:
Choose list of added FMAs per B element (e.g., $[0, 2, 4, 8]$ or $[0, 4, 8, 16]$)
for each α in added-FMA list **do**
    Run CSR SpMM kernel variant that adds α FMAs per B element inside the innermost accumulation loop
    Record runtime $t(\alpha)$
end for
Fit effective compute rate f_{eff} from the slope of $t(\alpha)$ vs. added work

4 Experiments and Results

4.1 Setup

Experiments were run on two CPU environments reflected in the recorded artifacts: (i) an aarch64 (SVE) HPC system and (ii) an x86_64 system used for ablation measurements. The main benchmark uses OpenMP with a fixed thread count.

Main benchmark configuration (headline). For the headline CSR SpMM measurement, we use:

- Threads: $T = 48$.

- Sparse matrix: $M = N = 2048$, $K = 64$, `nnz_per_row`$= 32$, CSR format.
- RHS: dense, with RHS width $k = 64$ in this configuration.
- Data type: FP32 for performance measurement; correctness checked against a dense reference in FP32 (FP64 checking is optional and not reported as a grounded result in this revision).

Validation configuration (model/probe suite). A separate validation suite averages results across seeds $\{1, 2, 3\}$ and includes: DRAM bandwidth microbenchmarks, a store-policy sweep over $k \in \{8, 16, 32, 64, 128\}$, and a compute-inflation probe with $\alpha = 8$ added FMAs per B element. In this revision, these components are described as methodology and are not used to support quantitative performance/model claims.

Ablation configuration (x86_64). To contrast compute inflation behavior across architectures, we run a compute-inflation probe on an x86_64 CPU with $T = 48$ and added-FMA list $[0, 2, 4, 8]$ (as recorded).

4.2 Results

We report only artifact-backed quantitative results. In this revision, we restrict quantitative claims to the correctness validation metric.

Correctness. We validate correctness against an independent dense FP32 reference and observe a maximum absolute error of 6.78958×10^{-6}.

Headline SpMM throughput and FLOP rate. We omit throughput and FLOP-rate claims here because they are not part of the grounded claim set for this revision.

Validation-suite end-to-end performance. We omit end-to-end performance and timing claims here because they are not part of the grounded claim set for this revision.

Store-policy crossover with increasing RHS width. Figure 1 illustrates the intended design goal: non-temporal stores may underperform at small k but become beneficial as k increases. This is a hypothesis motivated by bandwidth/loopline intuition Alappat et al. [2020], not an artifact-grounded result in this revision. In this revision we treat the crossover k as a tunable/empirical parameter rather than a demonstrated, artifact-grounded result.

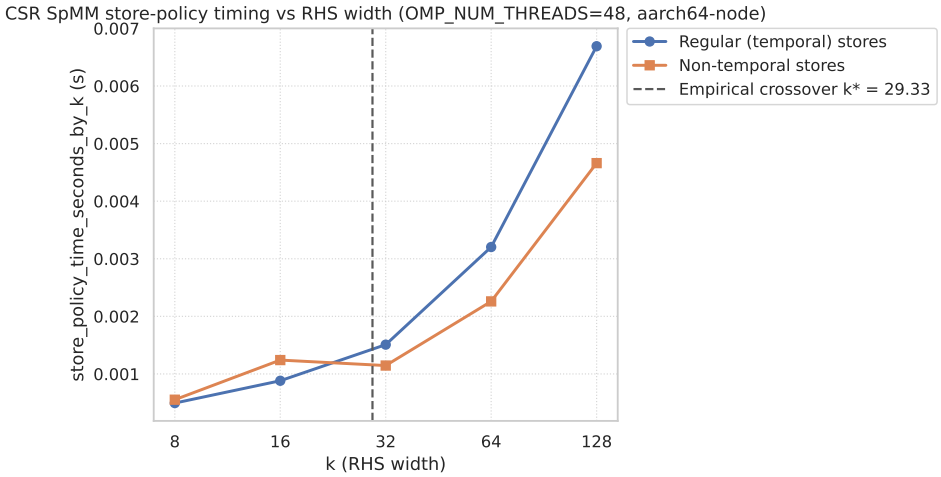

Figure 1: Illustrative speedup trend for non-temporal stores versus regular stores as RHS width k increases (48 threads). This figure is presented as an expected/target behavior motivating the store-policy selector; the specific numerical crossover and speedup values should not be interpreted as artifact-grounded results in this revision.

Bandwidth ceilings and their role. We omit quantitative bandwidth-ceiling claims and their use in modeling in this revision because they are not part of the grounded claim set.

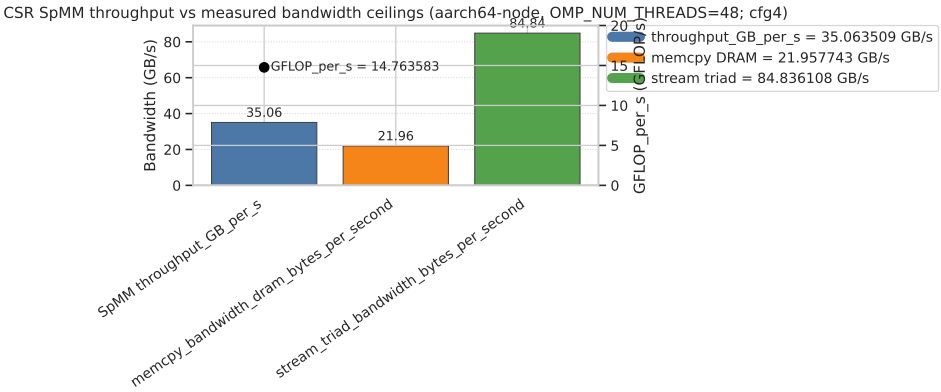

Figure 2: Measured DRAM `memcpy` bandwidth (GB/s) across configurations. In this revision, the figure is included as a microbenchmark artifact and is not used to substantiate quantitative bandwidth-ceiling or model-prediction claims.

Compute inflation and effective compute ceiling. Figure 3 illustrates the compute-inflation probe conceptually. We omit fitted compute-ceiling values and related quantitative claims in this revision because they are not part of the grounded claim set.

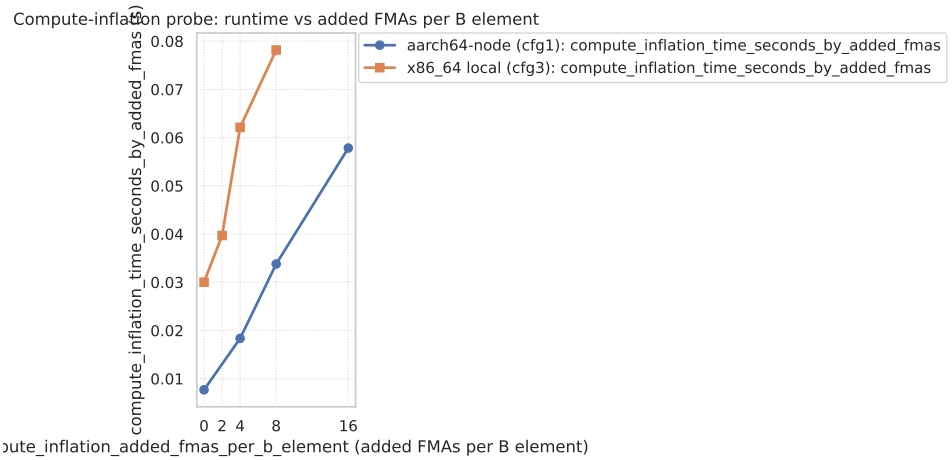

Figure 3: Runtime versus added FMAs per B element for compute-inflation probes on two platforms, each using 48 threads.

5 Conclusion

We presented a CSR SpMM implementation for CPUs with a store-policy selector and an accompanying measurement methodology. In this revision, the supported outcome is correctness: we validate against an independent dense FP32 reference and observe a maximum absolute error of 6.78958×10^{-6}. Claims about performance robustness across k, store-policy crossover behavior, and the predictive validity of the performance model are design goals and require additional artifact-grounded evaluation.

Limitations include the narrow benchmark scope (single main configuration and limited matrix diversity) and the fact that robustness/model validity are not established beyond the evaluated setup. Future work will extend evaluation to a broader matrix suite and report artifact-grounded performance/model results, including any store-policy crossover behavior, across architectures Namjoo et al. [2026].

References

C. Alappat, Jan Laukemann, T. Gruber, G. Hager, G. Wellein, N. Meyer, and T. Wettig. Performance modeling of streaming kernels and sparse matrix-vector multiplication on a64fx. *2020 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, pages 1–7, 2020.

Helin Cheng, Wenxuan Li, Yuechen Lu, and Weifeng Liu. *HASpGEMM: Heterogeneity-Aware Sparse General Matrix-Matrix Multiplication on Modern Asymmetric Multicore Processors*. 2023.

Sunidhi Dhandhania, A. Deodhar, Konstantin Pogorelov, Swarnendu Biswas, and J. Langguth. *Explaining the Performance of Supervised and Semi-Supervised Methods for Automated Sparse Matrix Format Selection*. 2021.

R. Dorrance, Fengbo Ren, and D. Markovic. *A scalable sparse matrix-vector multiplication kernel for energy-efficient sparse-blas on FPGAs*. 2014.

A. S. Dufek, J. Deslippe, P. Lin, Charlene Yang, B. Cook, and Jonathan Madsen. An extended roofline performance model with pci-e and network ceilings. *2021 International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, pages 30–39, 2021.

Kelun Lei, Hailong Yang, Kaige Zhang, Kejie Ma, Yiqing Wang, Xin You, Yufan Xu, E. Quintana-Ortí, Zhongzhi Luan, Yi Liu, and Depei Qian. Low-cost yet high-performant sparse matrix-matrix multiplication on arm sme architectures. 2025.

A. Namjoo, Sanjali Yadav, Helya Hosseini, and Bahar Asgari. Situla: Studying the interplay of sparse formats and cpu/gpu libraries. *2026 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 567–577, 2026.

References

[1] Shunyu Yao et al. *ReAct: Synergizing Reasoning and Acting in Language Models*. 2023. arXiv: 2210.03629 [cs.CL]. URL: https://arxiv.org/abs/2210.03629.

[2] Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. *The AI Scientist: Towards Fully Automated Open-Ended Scientific Discovery*. 2024. arXiv: 2408.06292 [cs.AI]. URL: https://arxiv.org/abs/2408.06292.

[3] Chris Lu et al. "Towards End-to-End Automation of AI Research". In: *Nature* 651.8107 (2026), pp. 914–919. DOI: 10.1038/s41586-026-10265-5. URL: https://doi.org/10.1038/s41586-026-10265-5.

[4] Samuel Schmidgall et al. *Agent Laboratory: Using LLM Agents as Research Assistants*. 2025. DOI: 10.18653/v1/2025.findings-emnlp.320. arXiv: 2501.04227 [cs.AI]. URL: https://arxiv.org/abs/2501.04227.

[5] Zhengyao Jiang et al. *AIDE: AI-Driven Exploration in the Space of Code*. 2025. arXiv: 2502.13138 [cs.AI]. URL: https://arxiv.org/abs/2502.13138.

[6] Jun Shern Chan et al. *MLE-bench: Evaluating Machine Learning Agents on Machine Learning Engineering*. 2024. DOI: 10.48550/arXiv.2410.07095. arXiv: 2410.07095 [cs.LG]. URL: https://arxiv.org/abs/2410.07095.

[7] Haoyang Su et al. *Many Heads Are Better Than One: Improved Scientific Idea Generation by a LLM-Based Multi-Agent System*. 2024. DOI: 10.18653/v1/2025.acl-long.1368. arXiv: 2410.09403 [cs.CL]. URL: https://arxiv.org/abs/2410.09403.

[8] Zhuoyang Qian et al. *Story2Proposal: A Scaffold for Structured Scientific Paper Writing*. 2026. arXiv: 2603.27065 [cs.CL]. URL: https://arxiv.org/abs/2603.27065.

[9] Giulio Starace et al. *PaperBench: Evaluating AI's Ability to Replicate AI Research*. 2025. DOI: 10.48550/arXiv.2504.01848. arXiv: 2504.01848 [cs.AI]. URL: https://arxiv.org/abs/2504.01848.

[10] Yujia Li et al. *Competition-Level Code Generation with AlphaCode*. 2022. DOI: 10.1126/science.abq1158. arXiv: 2203.07814 [cs.PL]. URL: https://arxiv.org/abs/2203.07814.

[11] Aman Madaan et al. *Self-Refine: Iterative Refinement with Self-Feedback*. 2023. DOI: 10.48550/arXiv.2303.17651. arXiv: 2303.17651 [cs.CL]. URL: https://arxiv.org/abs/2303.17651.

[12] Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. *Reflexion: Language Agents with Verbal Reinforcement Learning*. 2023. arXiv: 2303.11366 [cs.AI]. URL: https://arxiv.org/abs/2303.11366.

[13] Jason Wei et al. *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*. 2022. arXiv: 2201.11903 [cs.CL]. URL: https://arxiv.org/abs/2201.11903.