ARI: 自律型研究基盤

樹神 宇徳

v0.9.0, 2026-06-12

Abstract

本レポートでは、木探索ベースの探索ループと論文生成パイプラインを組み合わせた自律型研究基盤 **ARI** を述べる。探索ループは研究ステップノードに対する最良優先木探索 (BFTS) であり、ノード拡張は Reason-and-Act (ReAct [1]) 形式のエージェントが 14 のドメインスキルレジストリにツール呼び出しを発する形で行う。論文生成パイプラインは生き残った系譜から草稿を生成し、階層型ルブリックの下で反復的に改稿する。すべての成果物は単一のチェックポイントディレクトリに格納され、これが再現性、系譜、コスト計上の単位となる。本レポートはアルゴリズムと設計上の論拠を形式化し、予備的観察を報告し、設計を制約する決定性と新規性のトレードオフを議論する。

1 はじめに

1.1 動機

機械学習および周辺の計算分野における研究ワークフローは、共通の骨格を共有している: 研究アイディアをブレインストームし、実験を設計・実行し、結果を分析し、論文を執筆し、そして反復する。各ステップは大規模言語モデル (LLM) にとってますます扱いやすくなっており、発表済みのシステムは今やこのループの相当な区間を自動化している。AI Scientist [2, 3] は着想から論文までの全サイクルを閉じ、その結果を自動レビューアで採点する;Agent Laboratory [4] は人手で与えられた着想を LLM 判定のゲートのもとで実行する;AIDE [5] はエンジニアリングの内部ループを候補解上の木探索として定式化し、同一モデルを与えた条件下でこの定式化が汎用スキャフォルドを凌駕することを MLE-bench [6] が見出した;VirSci [7] は、マルチエージェントの熟議が単一エージェントよりも新規性の高い研究アイディアを生むことを示すが、アブストラクトで止まる;Story2Proposal [8] は、永続的な構造契約のもとで完成した実験的証拠を原稿へ変換する; そして PaperBench [9] は、エージェントが発表済み論文をどれだけ忠実に複製するかを事後的に計測する。

これらのシステムのいずれも拘束力をもって扱わないのは、生成された論文が**述べる**ことと、その実験が**生み出した**こととの関係である。存在する場合でも、検証は助言的にとどまる: AI Scientist のレビューアは完成稿を採点する一方、ハルシネーション制御はプロンプト水準にとどまる——その著者らは、誤って報告されたハードウェアと、改善として提示された悪化した指標を記録し、「報告された結果のより踏み込んだ自動検証」を重要な今後の課題として挙げている [2];Agent Laboratory の模擬レビューアは、10 点尺度で人間レビューアのスコアを 2.3 点過大評価する [4]; そして PaperBench の中心的な発見は実行ギャップである——エージェントは書かれたコードに対しては相当な得点を獲得するが、それを実行して論文の結果に一致させることに対してはほとんど得点しない [9]。

ARI はこのギャップを埋めるために存在する。ユーザは研究問題と計算予算を与える; システムは、発表可能な論文、各主張を裏付ける実験コード、そして到達経路の完全な監査記録を返す: 探索木の各ノード、各ツール呼び出し、各モデル呼び出し、そして API 利用の各ドルが、再現性の単位である単一のチェックポイントディレクトリ内に記録される。

1.2 リリース条件としての検証済み主張

ARI を際立たせる筋は、パイプライン全体を貫いて走る。着想の時点で、実行はメトリック契約を鋳造する: 最終的な論文が支えねばならない反証可能な主張であり、その各々が証拠とみなされる正確な測定名を明示する。LLM の命名は再生成を生き延びないため、契約は一度だけ鋳造され凍結される (section 4.3)。探索の間、ノードはそれらの名前のもとで測定を発し、その名前が未充足の主張へ向けて拡張を誘導する。そして系譜連鎖は、先行する測定から**計算された**証拠——あるフィット、その保留検証、モデルに基づく選択——を、繰り返しの probe へ退行する代わりに単一の系譜に沿って実行させる (section 3.8)。執筆の時点では、あらゆる数値への言及は、それが依拠するノードと測定に結びつけられ、決定論的な claim–evidence ゲート——ループ内に言語モデルは存在しない——が、報告された各数値を記録された結果から再導出する。客観的な虚偽は最終ゲートでリリースを阻止する; 補完的な意味論的レビューの主観的な所見は、改稿ループに助言するにとどまる。

1.3 貢献

本レポートは四つの貢献を形式化する:

1. **決定論的な claim–evidence ゲート**によって強制される **idea 所有のメトリック契約** (section 4.3): レビューアによる出力への意見ではなく、パイプラインの構造的性質としての検証。

2. 研究ステップノード上の**最良優先木探索** (BFTS) (section 3.1) であり、その選択と拡張は、構造化された候補記述上で LLM 主導であり、決定論的なフォールバックランキングを伴う (section 3.3)。

3. 階層型ルブリック R をキーに据える**論文生成パイプライン**; その葉判官が個々の主張を採点し、システムスコアは $\text{score}(P) = \sum_i w_i \, \text{judge}_i(P)$ となる (section 4.4)。PaperBench 複製器はツール呼び出しとして統合される (section 4.6)。

4. **チェックポイントスコープ**の規律: すべての外部状態 (メモリ、コスト台帳、系譜ポインタ) は `$ARI_CHECKPOINT_DIR` 配下に書かれ、ユーザのホームディレクトリには決して書き出されない。これは**決定性**という設計原則の運用上の帰結であり、section 2.5 が完全に提示する 5 つのうちの一つである (本報告では、そこおよび以降を通じてこの原則を P2 と呼ぶ)。

1.4 本レポートの適用範囲

本レポートは `ari-core` および 14 個の `ari-skill-*` パッケージの v0.9.0 を記述する。そのテーマは end-to-end の主張検証である:section 4.3 の mint-once 契約とゲートの強化; 系譜連鎖 (section 3.8); 発表済みの VirSci 熟議エンジンをライブの文献スナップショット上で無改変に駆動するオプトインの着想生成経路 (section 6.2); プラットフォームを認識した着想生成——探られたプラットフォーム能力の事実を着想生成に折り込み、計画が実行プラットフォームの提供しない測定を約束できないようにする; そして、それ自体がゲート検証済みであり、宣言された反証可能な主張 12 件のすべてが機械検証済みの証拠を伴うリリース済みのサンプル論文 (section 5)。これらの追加は v0.8.x の基盤の上に築かれる——プロトコルに忠実な三段の PaperBench 再現契約 (エージェントロールアウト、再現、採点)、ホスト真実の環境ガードレール、設定可能な探索評価層;section 5 で報告される非可逆圧縮対象に対するオペレーショナル再現パスは、当時最新の v0.8.0 パイプライン上で実施された。本文はアルゴリズムとデータフローをモジュール水準で記述し; システムが用いる逐語の LLM プロンプトは section B に列挙する。本版における経験的研究は予備的であり、制御されたベンチマークは今後の課題として残す。

1.5 構成

Section 2 は ARI のトップダウンビューを与える: オーケストレータ、14 個のスキル、パイプライン DAG、そして設計原則。Sections 3 and 4 は二つの技術的核であり、それぞれ独自の形式的定義をもつ;section 1.2 の主張検証の筋は両者にまたがって展開される。Section 5 は予備的観察を報告し;section 6 は本研究の位置づけを与え;section 7 は未解決問題を集約する。

2 システム概観

2.1 フルスタック

Figure 1 は v0.9.0 の ARI フルスタックを 5 層として描く; 本節はこれを上から下へ辿る。システムは一つのドライバと多数のツール提供者に分かれる: ドライバ (`ari-core`) はオーケストレータ、探索エンジンとエージェントエンジン、系譜歩行器、チェックポイント直列化器、コストトラッカ、設定ローダを所有し、一方で各ドメイン機能 — コードの記述、ベンチマークの実行、草稿のレビュー— はスキルの中に存在する。

入口。 実行はコマンドラインインタフェース (CLI)、公開 Python インタフェース (software development kit、SDK)、または可視化サーバ — ライブな探索状態、チェックポイント単位のファイル、起動コントロールをブラウザへ露出する HTTP ダッシュボード — のいずれかから始まる。すべての経路は同じオーケストレータ入口 (バッジ 1) へ収束する; 対話的な利用が第二のコードパスを得ることはない。

エンジン層。 *best-first tree search*(BFTS) エンジンは研究ステップノードの木を維持し、次にどのノードを実行しどう拡張するかを決定する (section 3)— これは ARI が AIDE と共有する、候補解の探索という枠組みである [5]。各拡張はノードを `AgentLoop` に渡す。これは Reason-and-Act(ReAct) エージェント [1] であり、ノードの作業が完了するまで自由テキストの推論とツール呼び出しを交互に織り込む: エンジンが方向を与え、エージェントが実行を担う — 一つの反復ループ (バッジ 2) である。ループの傍らの系譜歩行器は駆動されるのではなく参照される: これはチェックポイントを跨いで親ポインタを辿るため、派生実行は先祖のアイデアプールとコンテキストを継承できる (section 3.7)。

MCP 分配。 すべてのツール呼び出しは分配層 (バッジ 3) を横切る。*Model Context Protocol*(MCP) は、Anthropic が言語モデルエージェントループにツール的機能を露出させるために標準化した、JSON-RPC-over-stdio インタフェースである;ARI はこれを `ari-core` とすべてのスキルの間の汎用継ぎ目として用いる。各スキルは独立した Python パッケージであり、そのサーバはプールされた子プロセスとして — スキルが独自のインタプリタを同梱する場合はそのインタプリタの下で — 走る。スキルは、そのパッケージパスとツールが露出されるパイプラインフェーズを指定する設定エントリによって配線される; ツールマネージャはアクティブなフェーズでツールを濾過するため、exploration ノードが論文執筆ツールを呼び出すことはできない (section 3.6)。

現在ツリーに含まれる 14 のスキルパッケージは次の通り: `benchmark`、`coding`、`evaluator`、`hpc`、`idea`、`memory`、`orchestrator`、`paper`、`paper-re`、`plot`、`replicate`、`transform`、`vlm`、`web`。fig. 1 のレジストリはそれらのツール面をライフサイクルフェーズ別にまとめている: 生成 (idea、coding)、評価 (evaluator、review)、執筆および複製 (paper、replicate、paper-re)、そしてインフラと I-O — ここで `hpc` スキルはクラスタのバッチスケジューラへの継ぎ目であり、実験は投入ジョブとして走らせられる。注記すべきボックスが二つある:*review* はパッケージではなくツール面を指す (草稿レビューツールは evaluator と paper のスキルがホストする;section 4.5)。また `orchestrator` スキルは実行管理を MCP 越しに露出するため、ARI 自身を外部エージェントからツールとして駆動できる。

オーケストレータは呼び出しを分配するが、ドメイン作業自体は実行しない。この分離が重要なのは、探索アルゴリズムに触れずにスキルを追加・差し替えで

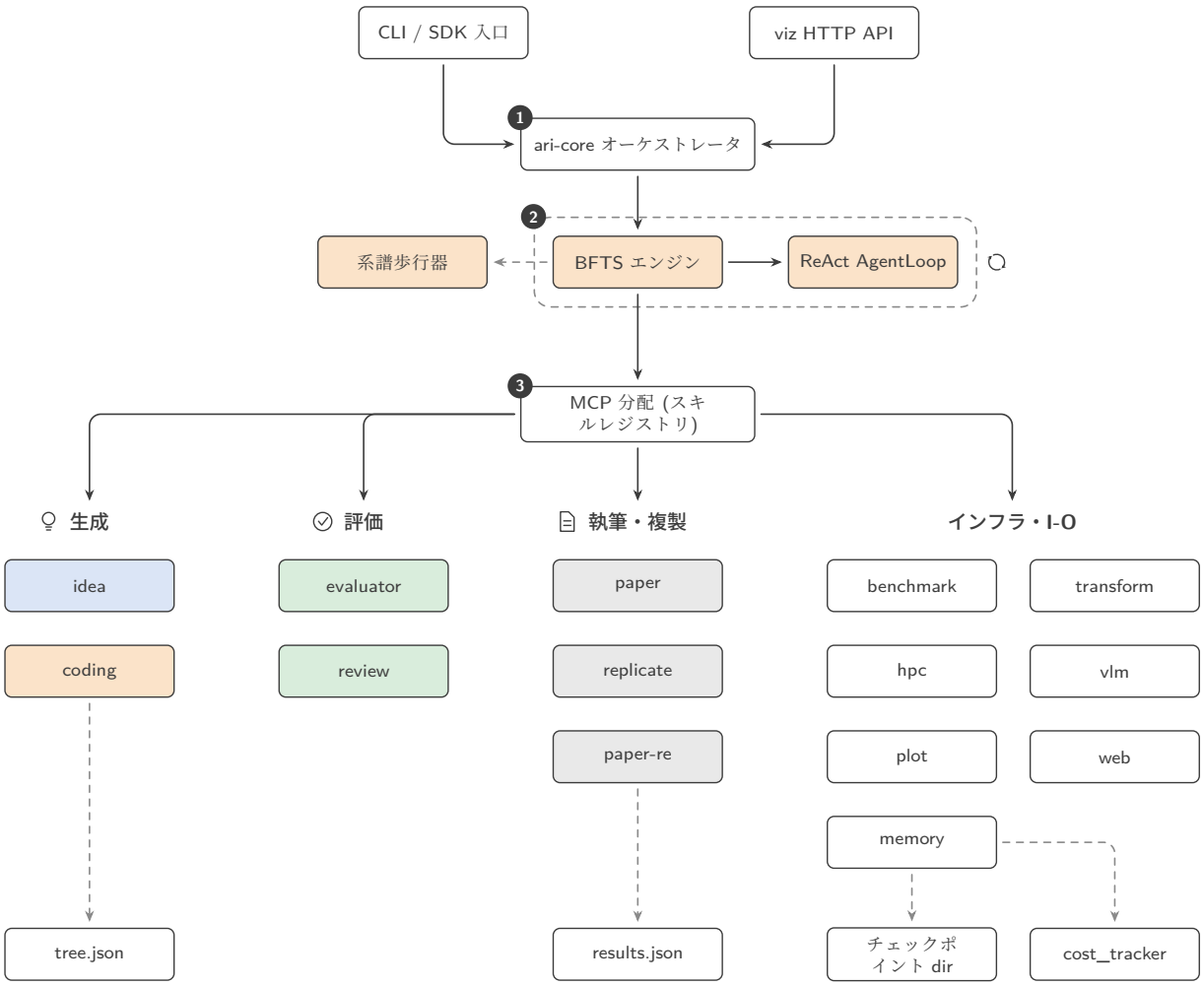

Figure 1: v0.9.0 における ARI のフルスタックを 5 層で示す: 入口; `ari-core` オーケストレータ (バッジ 1); エンジン層では BFTS エンジンと ReAct `AgentLoop` が反復ループ (バッジ 2) を成し、その傍らに系譜歩行器が置かれる;MCP 分配ハブ (バッジ 3) はスキルレジストリの 4 つのライフサイクル列へ扇状に広がる; そしてチェックポイント永続化。実線は制御、破線は副次的フロー (系譜参照、永続化) を表す。

きるからである:BFTS エンジンはどのスキルが参加するかを知らない。

最下層は永続化である: 実行が生み出すすべてのバイトは、直列化された木とコスト台帳とともに、単一のチェックポイントディレクトリへ落ちる (section 2.5)。fig. 2 の小さい図は本レポートの残りで使う「運用」ビュー (制御バスと状態エッジのみ) であり、fig. 1 は「層別」ビュー (ドライバ → エンジン → MCP → スキル → 永続化) である。

2.2 パイプライン DAG

一回の実行は 4 ステーションのパイプラインに帰着する (fig. 3)。idea ステーションは研究問題を選択または生成し;exploration ステーションは BFTS の下でノードを派生させ; paper ステーションは最良系譜を草稿に編成し;review ステーションは草稿をルブリックで採点する。グラフは DAG にレビューから paper への 1 本のバックエッジを加えた構造である: このエッジのみが非単調性を生じうる (改稿は軸を悪化させ得る)。そして 収束基準 (section 4.5) がバックエッジの回数に上界を与える。パイプラインはオーケストレータ内の単一の実行器が駆動する; 科学データ組立器が、執筆器の必要とするノード単位の成果物を照合する (section 4.2)。

write-and-review の帯は、草稿が最終化の前に通過しなければならない固定の**ゲート連鎖**を隠している。執筆後、草稿の主張は安定識別子によって記録済みの証拠に結び付けられる; 決定的な主張 – 証拠ゲート (言語モデルなし) が、報告されたすべての数値を記録済み結果から許容誤差内で再導出する; 補完的な意味論レビューが過剰主張を指摘して改稿ループへ供給する; そして改稿の後、結び付け・レビュー・ゲートが最終チェックとしてもう一度走り、厳格ポリシーの下では客観的な不一致が一つでも残る限り最終化を阻止する (section 4.3)。阻止ゲートは薄いスキルラッパー越しに到達される core 側のコードであるため、スキルの差し替えでこれを弱めることはできない; 阻止は客観的な虚偽に限って予約され、主観的なレビュー所見は改稿器を導くがけっして公開をゲートしない。この連鎖は Story2Proposal の契約を糸として通す生成を、実行に根ざした形で実現する [8]。

2.3 BFTS 制御フロー

Figure 4 は 1 BFTS ステップを端から端まで追う。実装は section 3 で展開する; 図は本レポートの残りで使う語彙 (フロンティア、選択、フォールバック、expand、prune、停滞) を確定し、LLM 主導の

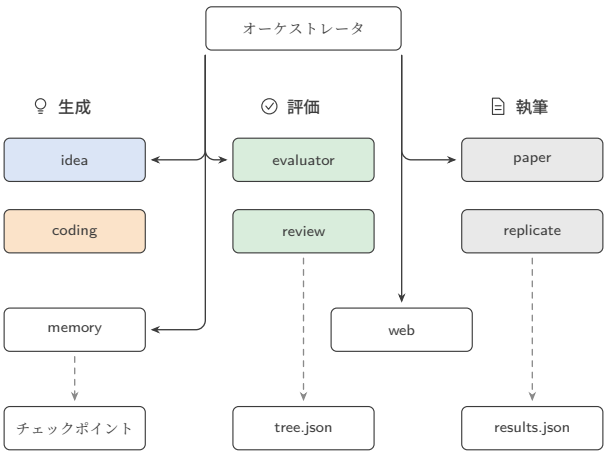

Figure 2: 運用ビュー。オーケストレータは二つの分配バスを通じて 3 つのライフサイクル列 (塗りがフェーズを符号化する) とフェーズに依らない支援スキルへ扇状に広がる。破線は状態を表す: 各生産グループは最下段のチェックポイント成果物へ永続化する。

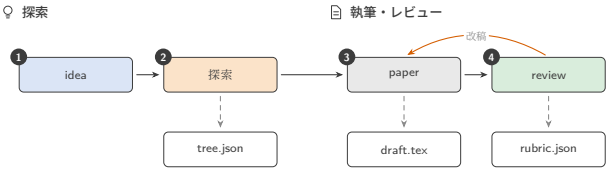

Figure 3: パイプライン DAG:*search* と *write & review* という二つの帯見出しの下に 4 つのステーション (バッジ 1–4) が並ぶ。各ステーションはその成果物をチェックポイントへ永続化する (破線)。唯一強調されたエッジが改稿 (refine) であり、草稿を書き直してレビューを再実行するバックエッジである。

二つのステーションの背後にあるプロンプトファイル (`bfts_select.md`、`bfts_expand.md`) を示す; 第三のプロンプト `bfts_expand_select.md` は、完全なループにおいて**どの**完了ノードを拡張するかの選択を駆動する (section 3.3)。背骨の下方にある二つのヒントボックスは v0.9.0 で新規であり、計算済み証拠のための系譜連鎖を実装する:coverage ヒントは拡張対象を、必要な測定値をすでに保持しているノードへと誘導し、lineage ヒントは新しい子に、どの継承測定ファイルを保持しているかを伝える。ヒントは名前のみを運び値はけっして運ばないため、枝の分離は保たれる (section 3.8)。

2.4 paper-re コンポーネントビュー

Figure 5 は `paper-re` — PaperBench プロトコルの複製経路 [9] を担うスキル — の二段階目のビューである。この層での ARI の貢献は小さいが具体的である: 薄いアダプタが各エージェントツールを包んでその出力をサイズ制限し、vendoring されたソルバのハードコードパスを ARI のノード単位ワークスペースへ振り向ける; それ以外はすべて PaperBench upstream を逐語的に流れ、これにより ARI は upstream の複製タスク枠組みと採点の意味論に整合したままとなる。section 4.6 は、bridge が駆動する rollout–reproduce–grade の 3 段階契約を展開する。

2.5 チェックポイントスコープと設計原則

ARI は 5 つの設計原則 (P1–P5) を中心に構築されている。最も拘束的なのが **P2: 決定性** である。すべての乱択ステップは乱数シードを記録し、すべての LLM 呼び出しはモデル同一性、プロンプト、出力を記録し、すべてのツール呼び出しはノード同一性でタグ付けされ (コピーオンライトのガード)、その書き込みが正しいノードの作業ツリーへ落ちることを保証する。チェックポイントを再走させれば、時計依存またはモデル側非決定性として明示的にマークされたフィールドを除き、バイト単位で同一の成果物を再現しなければならない。

その他の原則は、要約すると次の通り:

- **P1 単一成果物ツリー**: 実行のすべての出力は `$ARI_CHECKPOINT_DIR` 配下に存在する。`$HOME` や `/tmp` へ漏出しない。
- **P3 小コンポーネント**: 各スキルは独立したパッケージである; あるスキルを差し替えても `ari-core` を再構築する必要はない。
- **P4 明示的系譜**: 各ノードと各チェックポイントは親を記録するため、派生実験は差分のみを再計算できる。
- **P5 コスト透明性**: コストトラッカーが各モデル呼び出しにドルコストを付与し、利用者が実行を予算で制限できるようにする; スキルのサブプロセスも同じ実行レベルの台帳へ追記する。

チェックポイント直列化器は 3 つのトップレベルファイルを書き出す: 完全な探索木を `tree.json` に、論文パイプラインが消費する軽量なノード単位のエクスポートを `nodes_tree.json` に、実行末の集計を `results.json` に。チェックポイントを跨ぐ系譜歩行は、各チェックポイントがその実行メタデータに記録する親ポインタを通じて、子から先祖へと登る。

Figure 6 はチェックポイントのオンディスク形状を描く: 左に共有の実行レベルファイル、右にノード別作業ツリー。ノードの作業ツリーはその親のコピーとして始まるため、各ツリーはそのステップにおける実験の自己完結したスナップショットであり、証拠組立を根拠づける自己報告 (section 4.2) は、それが記述する成果物の隣に置かれる。この形状が契約である:ARI の出力を入力にとる任意のツール — 後続実行、リグレッションチェック、論文草稿 — は、このような単一のディレクトリを指せばよい。

3 探索アルゴリズム

探索フェーズは実行の実験的中核である: 選ばれたアイディアと、それが保持するメトリック契約 — 各々がその証拠となる正確な測定名を名指す、反証可能な主張の実行レベルの宣言 (section 4.3) — が与えられたとき、探索は有界なノード予算を、それらの主張を評価可能にする測定へと変換せねばならない。ARI はこのフェーズを研究ステップ上の最良優先木探索 (best-first tree search, BFTS) として組織する。骨格は AIDE に由来する。AIDE は LLM 駆動のエンジニアリングを、ハードコードされた draft/debug/improve 方針と単一

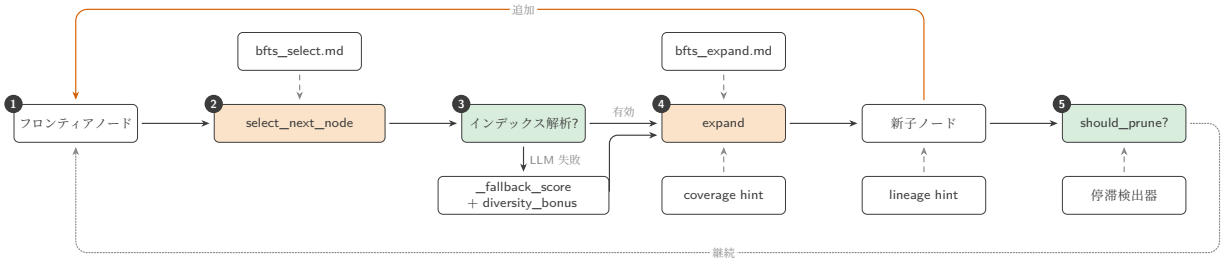

Figure 4: BFTS 制御フロー:1 ステップを、番号付き 5 ステーションの背骨として示す。LLM 主導の二つのステーションは各々が独自のプロンプトファイルを参照する (破線、上方); パースゲートが返されたインデックスを拒否したとき、選択は決定的な`_fallback_score`(scientific score に `diversity_bonus` を加えたもの) へフォールバックする。背骨の下方には、v0.9.0 の系譜連鎖ヒントが expand へ入り、停滞検出器が prune ゲートへ入る。強調されたエッジは子をフロンティアへ追加し、点線の余白経路は次の反復へ続く。

スカラー目的の下での候補解の木上の探索として枠づけた [5]。ARI はその方針を、決定的なフォールバックに支えられた LLM 主導の選択 (section 3.3) で置き換え、契約由来のカバレッジと系譜ヒントで拡張を操舵し (sections 3.4 and 3.8) — これは貪欲な単一目的方針について報告された局所最適のプラトー [5] への構造的な答えである — 各ノードを、その義務とフィードバックが同じ契約から来るガード付き Reason-and-Act(ReAct) エージェントとして実行する (section 3.6)。締めくくりの各小節は、探索が走る基盤を扱う: プラットフォームプローブ (section 3.9)、研究メモリ (section 3.10)、そしてルートを播種する構想段階 (section 3.11) である。

3.1 形式化

探索ループは木 $\mathcal{T} = (\mathcal{N}, E)$ を維持し、各ノード $n \in \mathcal{N}$ は研究の一ステップ — アイディア改訂、コード変更、ベンチマーク実行、分析など — を表す。ノードは以下を保持する:

- 離散状態 $s(n) \in \{\text{open}, \text{eval}, \text{done}, \text{failed}\}$、
- `NodeLabel` 列挙体から取られたカテゴリカルなラベル、
- 軸ごとの評価指標を保持する自由形式 dict `node.metrics`、区別されたスカラー `_scientific_score` $\in [0, 1]$ を含む、
- ノードが provisional なテキストではなく実測値を伴うかを示す Boolean `has_real_data`、ならびに
- 選択プロンプトおよび刈り込み判定で使われる帳簿フィールド `depth`。ARI は失敗ノードを再試行しない設計のため、`retry_count` シグナルは一切提示されない。

Figure 7 はこの種の小さな木を示す。軸レベル信号から scientific score への集約は、`Evaluator` プロトコルの任意の実装に委譲される;ARI は固定の線形結合を pin しない。プロトコルは BFTS と下流のルブリック評価との唯一の継ぎ目であり、軸をスカラーへ縮約する方法について検索エンジンを agnostic に保つ。

Run-loop 状態. 外側ループの各反復は次のタプルを変換する:

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

ここで $\mathcal{F} \subseteq \mathcal{N}$ は拡張可能な完了ノード集合 (done / failed), $\mathcal{P} \subseteq \mathcal{N}$ は実行待ち (open) ノード集合, $e : \mathcal{N} \to \mathbb{N}$ は各フロンティアノードが何回展開されたかをカウント, $H = (l_1, \ldots, l_t)$ は実行済みラベルのスライディングウィンドウ (長さ上限 $W = 20$、section 3.3 参照) である。初期状態は $S_0 = (\emptyset, \{n_{\text{root}}\}, \mathbf{0}, ())$。

枝刈り述語. ハードな探索予算は次の述語で表される:

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

ここで B_N = `config.max_total_nodes`, B_D = `config.max_depth`, sterile 述語

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

(Δ_n は n が追加・変更・削除したファイルの集合) は親に対して 1 ファイルも新規アーティファクトを書き出さずに終了したノードで真となる。run-loop がこのファイル差分を計算し、結果を Boolean `_sterile` フラグとしてノードに記録する;`should_prune` はこのフラグと 2 つの上限を読むだけである。呼び出し側が毎反復 live な $|\mathcal{N}|$ を渡すことで, ノード数の単一の真理源が保たれる。ステップ・コスト予算は BFTS エンジン内ではなく呼び出し側のコストトラッカーによって別途強制される。

Retire 述語. Π に加え、run-loop はより柔らかい **フロンティア** *retire* **述語** ρ を適用し、最も生産的なリードでなくなった親 f を $\mathcal{F}$ から除去する:

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

ただし $E_{\max}$ = `config.max_expansions_per_node` (既定値 4)。規則 A は子 c が親 f を上回った瞬間に f を retire し、規則 B は親ごとの予算を制限して探索を広げる。ρ はノードを**削除しない** — その履歴は $\mathcal{T}$ に残る — 再展開対象から外すだけである。

外側ループ 1 反復の段階分解 (内部展開サブループ、並列 ReAct バッチ、実行後 retire) は fig. 8 に示す。明示的な擬似コードは section 3.2 の algorithm 1 を参照。

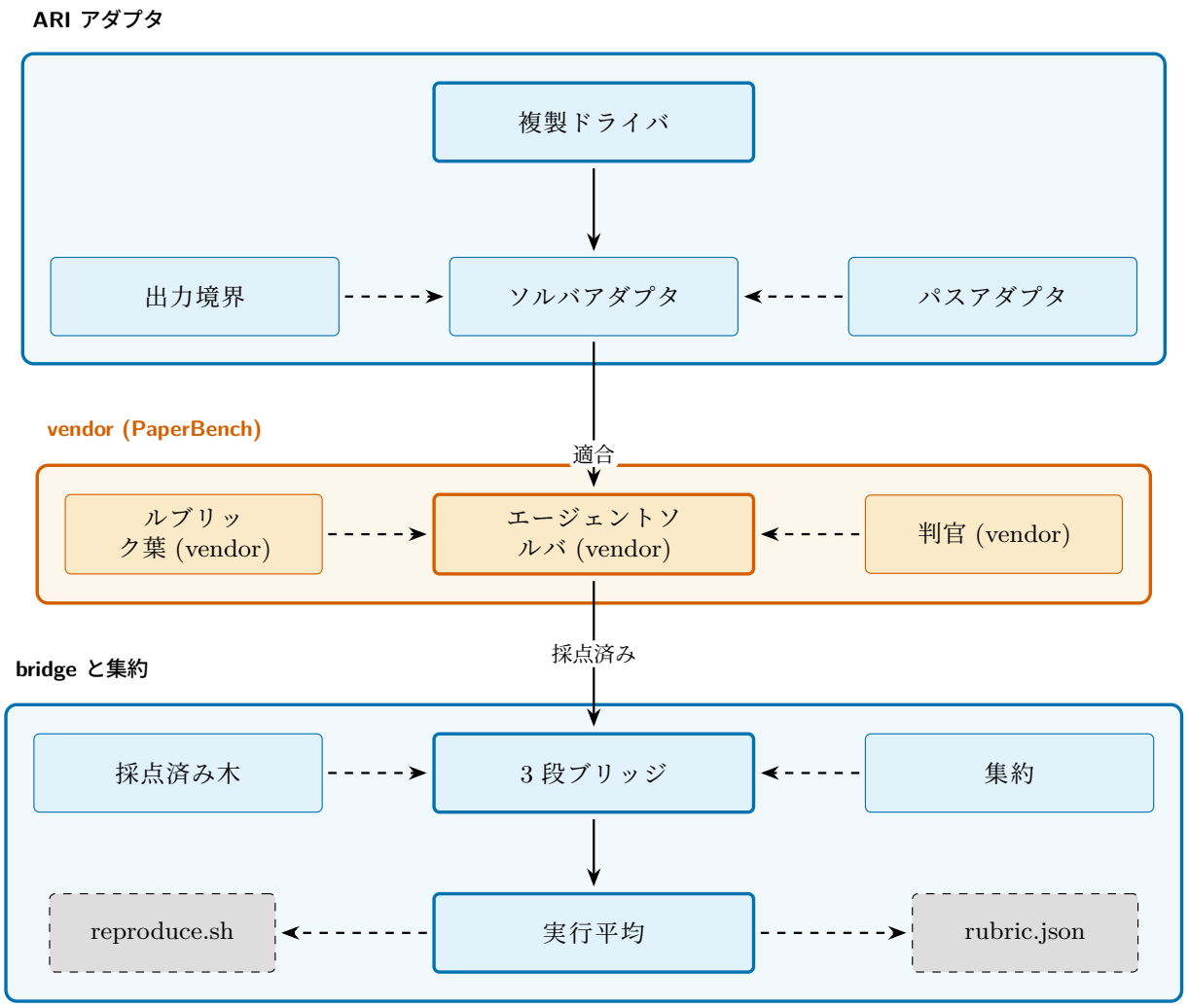

Figure 5: `paper-re` スキルの構成要素を 3 層で示す。最上段は ARI のアダプタ層 (複製ドライバ、ソルバアダプタ、出力境界、パスアダプタ); 中段は vendoring された PaperBench パッケージ (エージェントソルバ、ルブリック葉、判官); 最下段は、3 つのプロトコル段階を駆動し、submission ごとの採点済み木 — 反復実行を平均化して — を単一の `rubric.json` スコアと ARI 側の `reproduce.sh` に畳み込む bridge である。詳細は section 4.6。

3.2 Run-loop アルゴリズム

Algorithm 1 は反復を明示的な擬似コードとして書き、fig. 4 は同じステップをプロンプトファイル込みの制御フロー背骨として描く。内側 WHILE ループは空きワーカースロットを 1 展開ずつ埋め (ワーカーは次の展開提案前に必ず走り始める)、外側ブロックは `select_next_node` を 1 回 1 ノードずつ繰り返し呼んでワーカー上限までバッチを組み立て、そのバッチを並列実行する。実行後ブロックは (a) 実行されたラベルを H に追記して多様性ボーナスを実態に整合させ、(b) ρ の規則 A / B を適用する。

3.3 ノード選択 (LLM 主導)

ARI は意図的にノードのランキングを閉形式のスカラー効用に **縮約しない** — これは ARI の BFTS を AIDE のハードコードされた貪欲方針 [5] から最も分かつ設計上の選択である。代わりに、`select_next_node` (「次のエージェントステップが実行すべき pending ノードはどれか?」) と `select_best_to_expand`(「拡張するに最も値する完了ノードはどれか?」) の双方が候補記述リストを LLM に渡し、0 始まりの単一インデックスを解析して受け取る — 各呼び出しはちょうど 1 ノードを選ぶ。各ノード n の候補記述 (fig. 9) は次の形式の一行:

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

実行選択側 (`select_next_node`) は、現在ボーナス対象のノードに対してこの行末へ `diversity_bonus=+0.05` を付加する; `select_best_to_expand` は付加しない。これを消費するプロンプトは section B.3(選択) および section B.2(拡張対象) に逐語掲載する。プロンプトが列挙する選択基準は: `has_real_data=True` で強い metrics を持つノードは高価値;metrics は全体論的に多目的に重み付け; 優れた結果を持つ深いノードは継続に値する;`diversity_bonus` はソフトなタイブレーカとしてのみ扱う、である。`label` フィールドは `select_best_to_expand` に示される情報と一致しており、誤解を招く「low retry を優先」基準は存在しない。

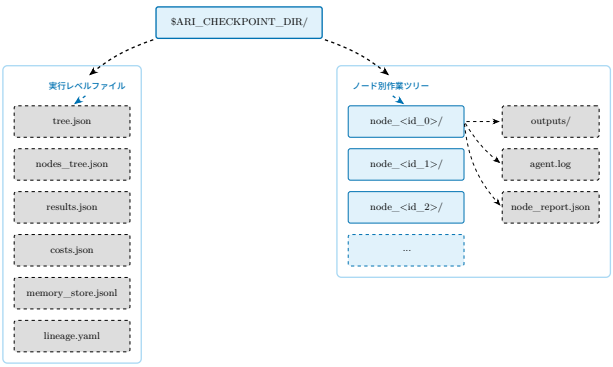

Figure 6: チェックポイントディレクトリのレイアウト。実行レベルファイル (左帯) はオーケストレータが書く; ノード別作業ツリー (右帯) は 1 BFTS 拡張内で生成された成果物を保持し、その中には各ノードが残す構造化された自己報告も含まれる。実行の再現はまさに「このディレクトリに対して再実行する」ことであり、レイアウトはそれ以外の状態が不要であることを保証する。

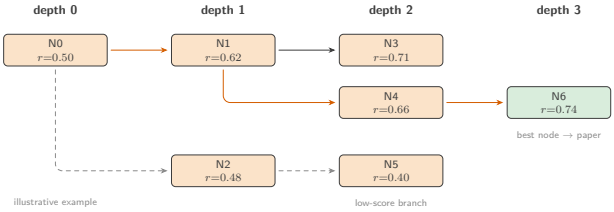

Figure 7: 小さな BFTS 探索木 (合成・例示用)。列は探索深さ; 各ボックスはスカラー報酬 $r(\cdot)$ を持つ 1 ノードである。強調された辺は最良ノードの系譜をルートまで辿る—論文フェーズが消費する連鎖 (section 4.2) である; 破線の辺は選択方針が以降選ばなくなる低スコア分岐を示し、その履歴は木上に残る。

多様性ボーナス

多様性ボーナス `BFTS.diversity_bonus` は最近のラベルを`_recent_label_history`(`record_run` で埋まるスライディング窓) で追跡する。当該窓内で最頻ラベルに対し相対的に**過少表現**のラベルを持つノードに対し、関数は次を返す:

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

ただし $\mathrm{cnt}(n)$ は窓内における $\mathrm{label}(n)$ の出現回数で,$\mathrm{cnt}_{\max} = \max_{\ell} \mathrm{cnt}(\ell)$ である。0.05 という定数は意図的に小さい:scientific score が依然としてランキングを支配し、ボーナスはほぼ同点のものを破るのみである。

LLM フォールバック

LLM が解析不能なインデックスを返した場合、`select_next_node` は `_select_fallback` ヘルパ (候補を `_fallback_score` 式で順位付けする) による決定的なランキングへフォールバックする。デフォルトの式は

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

ここで $r(n) \equiv$ `_scientific_score`(n) はノードのスカラー報酬 (section A) である。`_select_fallback` は

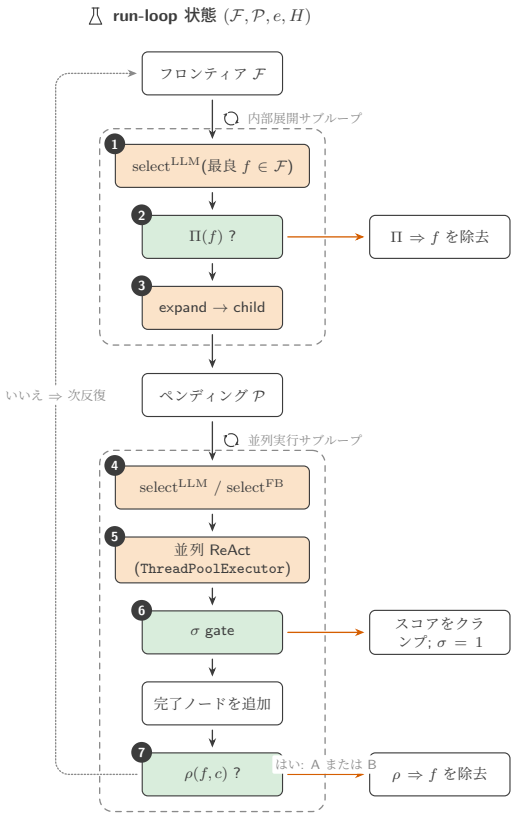

Figure 8: BFTS 外側ループ 1 反復の状態機械ビュー (上から下への単一フローチャート)。ステップ 1–3 が内部展開サブループ、ステップ 4–7 が外側並列実行ループ。黄色のひし形は述語ガード (Π, σ, ρ)、右側の赤破線ボックスは各述語が起こす副作用 (フロンティア除去 / スコアクランプ / retire)。図中に描かない補助状態: 親ごとの展開カウンタ $e(\cdot)$ はステップ 3 で増分されステップ 7 の ρ で参照される。ラベル履歴 H(窓 W) はステップ 5 の `record_run` で追記され、ステップ 4 で多様性ボーナス $\mathrm{div}(\cdot)$ の計算に用いられる。fig. 4(同じループを LLM プロンプト込みの細粒度制御フローとして示す) と比較されたい。

`has_real_data=True` のノードがあればそれを優先し、次いで $\mathrm{fb}(\cdot)$ 最大の候補を返す。これにより、LLM 呼び出しが degrade してもループは必ず前進する。

評価層の設定

sections 3.1 and 3.3 で登場する 4 つの評価面は独立に設定可能である (デフォルト値は上式を再現するため、未変更の設定は no-op となる)。

合成式. `evaluator.composite` で指定。$\{a_k\} \to$ `_scientific_score` の縮約として、加重調和平均 (デフォルト)、加重算術平均、ボトルネック型の `weighted_min`、加重幾何平均から選択。調和・幾何平均は分母を有限に保つために $\epsilon = 0.01$ で 0 軸をフロアする。

フロンティアスコア戦略. `bfts.frontier_score` で指定。eq. (5) は `scientific_plus_diversity` に相当。他の選択肢は `scientific_only` (div を落とす);`depth_penalized` は $\mathrm{div}(n)$ に $-\lambda\,\mathrm{depth}(n)$ を加える ($\lambda =$`depth_penalty_lambda`); `ucb_like` は $\mathrm{div}(n)$ に $c_{\mathrm{ucb}}\sqrt{\log N/(e(n)+1)}$ を加える ($c_{\mathrm{ucb}} =$`ucb_c`、$N = \sum_{n'} e(n') + |\mathcal{F}|$)。

Algorithm 1 BFTS run-loop(外側ループ 1 反復)。

Input: 状態 $S = (\mathcal{F}, \mathcal{P}, e, H)$; 設定 $(B_N, B_D, E_{\max})$ とワーカー上限 `max_parallel` $= \min(\texttt{max\_parallel\_nodes}, 4)$
Output: 更新後の状態 S'

▷ — 内部展開サブループ —
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
▷ — 外側並列実行バッチ —
$B \leftarrow \emptyset$
while $|B| < \min(\texttt{max\_parallel}, |\mathcal{P}|)$ **do**
  $n \leftarrow \text{select}^{\text{LLM}}(\mathcal{P})$ ▷ `select_next_node`: 1 ノード
  $\mathcal{P} \leftarrow \mathcal{P} \setminus \{n\}$; $B \leftarrow B \cup \{n\}$
end while
for all $n \in B$ **並列 do**
  $n' \leftarrow \text{ReAct}(n)$ ▷ `AgentLoop.run`、失敗あり
  $H \leftarrow (H \,\|\, \text{label}(n'))[-W:]$ ▷ `record_run`
  $\mathcal{F} \leftarrow \mathcal{F} \cup \{n'\}$
  for all $p \in \mathcal{F} : p = \text{pa}(n')$ **do** ▷ 規則 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** ▷ 規則 B
    if $e(p) \geq E_{\max}$ **then**
      $\mathcal{F} \leftarrow \mathcal{F} \setminus \{p\}$
    end if
  end for
end for
return $S' = (\mathcal{F}, \mathcal{P}, e, H)$

軸セット. `evaluator.axis_mode` で指定。`dynamic`(デフォルト) は汎用 5 軸フロア + 有効 rubric + `idea.json` のプランキーワードから自動構築。`legacy` は 5 軸固定。`custom` は $\{(\text{name}, \text{description}, w)\}$ のリストをそのまま judge LLM に渡す。

選択プロンプト. `bfts.select_prompt` および `bfts.expand_select_prompt` で指定。algorithm 1 の 2 つの $\text{select}^{\text{LLM}}$ 呼び出しで使うテンプレートを、`FilesystemPromptLoader` キーで差し替え可能。selection テンプレートは `experiment_goal`、`memory_context`、`candidates` の placeholder を受け取り、expansion-target テンプレートは `experiment_goal` と `candidates` を受け取る。

3.4 拡張 (1 呼び出しにつき 1 子)

`expand` 関数は 1 回の呼び出しで**高々 *1* つ**の新しい子を生成する (事前バッチ化なし)。同じ親に対して複数の子が欲しい呼び出し側は `expand` を反復的に呼び、これまでに生成された子を `existing_children` 引数経由で渡すことで、LLM が重複方向を提案するのを避ける。

各新子のラベルはハードコードされたテンプレートではなく LLM の判断で決まる; 拡張プロンプト (section B.1 に逐語掲載) には以下が渡される:

- 親の状態 (`succeeded` / `failed/no-real-data`) と `_scientific_score`;
- 親の構造化された `node_report` (`delta_vs_parent` / `concerns` / `hints`、node-report ビルダが生成) が存在する場合、planner が特定の弱点を狙えるよう含める;
- 同じ深さの兄弟スコア、ならびにルートから親までの先祖スコア;
- ツリー全体の多様性指標 (これまでに見たユニークラベル、深さ分布); ならびに
- この親で既に生成された子のラベル (重複提案を避けるため)。

子は open で生成され、エージェントループに実行のため渡され (section 3.6)、すべての軸が埋まれば done に、エージェントループが例外を投げれば failed に遷移する。

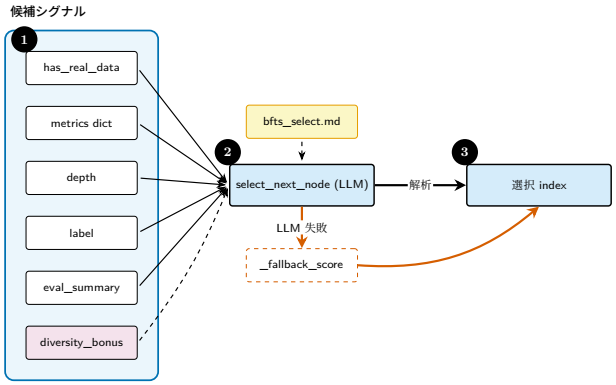

Figure 9: 次に操作するノードを選ぶ LLM に渡される選択シグナル。入力は構造化された候補記述に連結される; ソフトな `diversity_bonus` はタイブレーカであり、主シグナルではない。

カバレッジ操舵. 拡張対象セレクタは本質的にブランチを跨ぐスケジューラ — 各ブランチのスコア、metrics、要約のすべてを見る — であるが、スコアだけでは実行が宣言した主張について何も語らず、ある実際の実行では 10 ノードの木が同じ目玉実験の 10 通りの変種を生む一方で、宣言された機構の主張には専用の実験が一つも当てられなかった。そこで `select_best_to_expand` に渡される目標テキストは、メトリック契約から導かれる実行レベルの 主張カバレッジブロックで拡張される: どの宣言済み主張が実行のどこかですでに裏づけ測定を持ち、どれが未カバーのままかを示し、「未カバーの主張をカバーする」が**どの**ノードを拡張するかを左右できるようにする。ある主張が要求する測定名はその 契約証拠名である。カバレッジは二点で保守的に計算される: ある主張がカバー済みとされるのは、その要求する名前の**過半数**が出力されたときに限る — これは最終ゲートの任意名一致規則よりも意図的に厳しい。操舵においては偶然に共有された一つの汎用名が専用実験を永遠に抑止しかねない一方、未カバーと誤

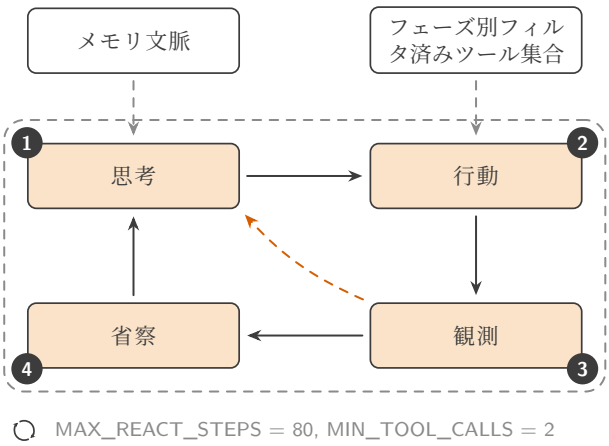

Figure 10: ReAct ループ。実線が主サイクル; 破線のバックエッジは観測がすでに開いた問いを閉じた場合の早期離脱を表す。

る方は高々冗長な測定を冒すだけだからである — そして、評価器が実データありと認めたノードのみが寄与し、ゲートと整合する。これにより、欠陥のあるブランチの単なる測定**名**が兄弟に「すでにカバー済み」と告げることはできない。同じ状態はノードごとに再計算され、実行中の各エージェントにも届く (section 3.6)。**計算された**証拠を単一の系譜に沿って連鎖させる対のヒントは section 3.8 の主題である。

3.5 枝刈りと停止

`should_prune` は枝刈り述語 Π(eq. (1)) を実装する: 総ノード上限、深さ上限 (`max_depth` 設定フィールド)、および sterile フラグである。フロンティア retire 述語 ρ(eq. (3)) は各ノードの完了後に適用される: ノードを削除せず、その履歴は $\mathcal{T}$ に残るが、フロンティアのプールが縮むため、同じ親を無限に掘り続けるのではなく探索が広がる。

ループは次のいずれかが満たされれば停止する:

- グローバル `max_total_nodes` 上限到達;
- フロンティアの**すべての**ノードが Π に該当 (深さ上限、sterile、または予算枯渇) し pending プールも空である;
- コストトラッカーが強制する呼び出しごとのステップ / コスト予算の枯渇;
- 停滞検出器 `detect_stagnation` が、`_scientific_score` の移動最大値がスライディング窓内で改善しないことを観測;
- 系譜判定モジュールからの `LineageDecision` が当該分岐を明示的に停止 (section B.4 参照)。

これらのうちどれが実際に最も頻出するかは経験的問いであり、凍結チェックポイントが揃い次第 section 5 で答える; 本レポートはその分布を主張しない。

3.6 ReAct ドライバ

各ノードの拡張は `AgentLoop` クラスの ReAct エージェントループ [1] を走らせる (fig. 10): モデルは蓄積された文脈について推論し、ツール呼び出しを発行し、構造化された結果を観測し、これを繰り返す。最大ステップ数は `MAX_REACT_STEPS = 80` で、インスタンスごとに `max_react_steps` キーワードで上書きできる。別ガード `MIN_TOOL_CALLS = 2` が trivially 短い軌跡を防ぐ。エージェントのシステムプロンプトは section B.6 に逐語掲載する。

エージェントが利用できるツール集合はツールマネージャがフェーズ別にフィルタする; 例えば探索フェーズは論文執筆ツールをマスクし、BFTS 拡張が草稿にショートカットされるのを防ぐ。一部の MCP ツールは `ari-core` 自身用に予約されており (memory CoW 操作)、LLM からは隠されるため、モデルが任意のノード ID を設定してメモリスキルのコピーオンライトチェックを迂回することはできない。

ノード開始時の義務

ノードは空白のプロンプトから始まるわけではない。最初のモデル呼び出しの前に、ループは決定的な文脈を注入する:section 3.10 の 作業文脈 — 実験の固定核とその先祖の結論 — と、実行のメトリック契約が存在するようになれば、ドメイン中立な言葉で、最終的な主張が単に尤もらしいのではなく科学的に妥当であるためにノードが何をせねばならないかを述べる 契約義務メッセージである: 目玉数値が用いる問題サイズで、独立した参照に対し正しさを検証し、残差を測定として記録する; 正規化の分母に使われる天井は決して**ハードコードせず** — **測定する**; 各天井と正しさ検査の来歴をタグ付けし、それらが実行されたことをゲートが確認できるようにする; 目玉メトリックがその生のオペランドからどう再計算されるかを宣言する; そして各宣言済み主張を評価可能にする測定を、**正確な**宣言名のもとで出力する — 否定的な結果は許容されるが、証拠のない主張は許容されない。このテキストはドメイン知識を一切持たない; エージェントは自らのドメインが要請するとおりに各義務を果たす (スループット天井ならマイクロベンチマーク、精度天井ならベースライン実行、数値法なら解析的検査)。

実行レベルの付録 3 つが同じメッセージに同乗する: ノードの主張カバレッジ状態 (section 3.4) は、実現可能に測定できる未カバー主張を 1 つか 2 つ優先するよう求める; section 3.8 の継承データノート; および section 3.9 のプラットフォームノートである。これらの実行レベル不変量は *pin* される: スライディング文脈窓はこれらを — survey、アイディア生成、metric-spec ツール結果とともに — 常に保持し、古いターンは脱落する。ルートでは、文脈構築時に契約がまだ存在しないため、義務は契約鋳造ツール呼び出しのハンドラがループ途中で注入する。こうしてどのノードも宣言済み主張を知らずに走ることはなく、義務を二度受け取るノードもない。

出力フィードバック

測定は単一の構造化出力ツール (`emit_results`) を通じてノードを離れる。これはノードの結果ファイルを書き、メトリック契約に対し検証し、まだ欠けている必須証拠を名指す 契約警告を返す。初期の実行はまさにこの地点で失敗の形を露呈した: 警告はツール結果の内部に届き、ハーネスは最初の完了ジョブの直後にノー

ドを強制終了し、満たされない義務は行動するステップを残されなかった — 観測されたあるノードは 80 ステップのうち 66 ステップを未使用のまま終わった。そこでループは出力時点のフィードバックを実行可能なターンへ変える: 出力が契約警告を伴うとき、継続を促すナッジ — 欠けている測定、残りステップ、そしてそれらを今すぐ実行して再出力せよ (再出力は上書きする)、あるいはこのノードでは実現不能な場合のみ結論せよ、という指示 — を注入し、ノードを次に述べる強制終了ガードを抑止する契約ホールドに置く。ホールドはステップ上限の 10 ステップ前に失効し、ドゥームループ防止のバックストップを保ち、警告のない出力がこれを解除する。

強制終了ガード

ループは軌跡の両端を守る。早すぎる成功宣言に対しては、ノードは `MIN_TOOL_CALLS` 回のツール呼び出し前には終了できず、—そのツールセットに実行ツールが含まれるときは常に — 何かを実行してその出力を読む前には終了できない。ReAct エージェントで記録された反復ループの失敗モード (元の研究は推論失敗の大きな割合を、再発行されたほぼ同一の行動に帰している [1]) に対しては、ループが 強制終了する: ノードが出力を読まれた完了ジョブを持ち、少なくとも 5 ステップが経過し、契約ホールドが有効でなくなれば — あるいは最小ガードを満たした上で上限直前の最後の 3 ステップでは — ツール呼び出し要件が外され、モデルの次の出力はその最終構造化レポートでなければならなくなる。反復はツールレベルでも減衰される (アイディア生成ツールは最初の成功呼び出しの後に抑止される) し、注入されたユーザメッセージはメッセージブロック境界へ遅延されるため、再生される会話がツール呼び出しをその応答から分割することはない。

3.7 系譜と再現性

系譜とは最良ノードの先祖の連鎖であり、`tree.json` の `lineage` キーとしてチェックポイントへ書かれる。`LineageState` データクラスは BFTS が継続判定に用いる移動統計を保持し、跨チェックポイントの歩行器 `walk_ancestor_ckpts` は実行を跨ぐ親ポインタを提供する。アイディアプールは `get_idea_pool_for_ckpt` 経由で継承され、ランキング付きルートアイディアの選択は `RootChoice` データクラスに記録されて事後監査可能となる (プロンプト: section B.5)。

再現性は三つの不変量にかかる。第一に、すべての乱択操作はノード識別子から自身を播種する。第二に、すべてのツール呼び出しは引数と出力を該当ノードの作業ツリーへ書き、決して親へは書かない —これは MCP 層の `cow_node_id` チェックで強制される。第三に、コスト台帳が `CostTracker.init` でノードあたりのドル額を付与するため、同一プロンプトでの予算検査は決定的となる。

既定はオフライン。 再現性は探索ループが何を読めるかも規定する。既定では各ノードのエージェントループは**オフライン**で走る — Web アクセススキルは論文執筆と再現のフェーズにゲートされており、ノードは探索の最中に時々刻々変わるライブ検索結果を参照できないため、ノードごとの軌跡はその日の Web の状態に依存しない。文献調査は依然行われるが、より早く枠の外で — 探索開始前、構想時に行う有界な調査として行われる (section 3.11)。ライブ情報が真に必要な実行は、オプトイン (`bfts.allow_web`) で探索中のノードエージェントへ Web アクセスを開放できる; その場合、実行はチェックポイントに非再現軌跡マーカーを書き込むため、後の再現が暗黙に厳密扱いされることはない。いずれにせよ決定性の契約は誠実に保たれる: 既定の実行はバイト単位で再現可能であり、再現性をライブ情報と引き換えにする実行は、その取引を自身の来歴に記録する。

3.8 計算された証拠のための系譜連鎖

実行が支えようとする主張のすべてが、新規に測定されるわけではない。一部の証拠は、すでに存在する測定から**計算**される —先行する probe 群に対するパラメータフィット、そのフィットの held-out 検証、構成間のモデルに基づく選択である。こうした主張は、互いに独立な木ノードでは構造的に到達不能であることが判明した: 実際の実行では、フィッティングや検証に向けられた子であっても、測定を持たない親から拡張されると単発 probe の再実行へ退行した。「すでに手元にあるデータから計算する」が可視の選択肢として一度も現れない — 子はデータを持たず、どのノードが持つかをセレクタに教えるものが何もない — ためである。

この間隙を塞ぐ機構は、木が既に持つ親 → 子の作業ディレクトリ継承 (子は親の作業ツリーの複製内で実行される; section 4.2 も同じ事実に依拠する) のみを、拡張の両側に 1 つずつのヒントとして用いる。親側では、拡張対象選択の目標に付加される主張カバレッジブロック (section 3.4) が、契約証拠名を作業ツリーに最も多く保持するノードを名指す系譜ヒントを得るため、フィットや検証を担う子は、その入力を既に保持するノードから優先的に拡張される。子側では、継承したディレクトリに系譜上の測定ファイルが含まれる子に対し、開始時 (section 3.6 の継承データノート) に、どのファイルが存在しどの正確な契約名を含むかが伝えられ、基礎にある実験を再実行するのではなく、それらのファイルから導出証拠を**計算する**よう指示される。

ノード境界を越えるのは名前だけである: ファイル名と測定名であり、値も、兄弟ノードの結論も、決して越えない。これらのヒントは契約自体と同種の実行レベルの調整メタデータであるため、section 3.10 のブランチ隔離は保たれ — 欠陥のあるブランチの数値が兄弟の推論へ漏れることは依然としてできない —その一方で、計算された証拠の連鎖 (フィット、次いで検証、次いで選択) は、繰り返しの probe へ潰れる代わりに単一の系譜に沿って実行できる。

3.9 プラットフォームプローブ

研究計画は、その科学について誤る前に、そのプラットフォームについて誤りうる。ある実際の実行では宣言済み主張がハードウェアカウンタのプロファイルを要求したが、想定されたプロファイラは実験が走った計算区画から検証可能なほど明らかに不在だった: それらの主張は恒久的に充足不能であり、最終ゲートは — 正しく — どのノードも決して生み出せない証拠ゆ

えに論文をブロックした。したがってプラットフォームの実現可能性は、想定されるのではなく**測定**されねばならない— しかも、下流の何かが測定語彙にコミットする前に、一度測定されねばならない。

実行がバッチ区画とともに設定されているとき、実行開始は一度きりのプラットフォームプローブを発行する: 計算区画自体で実行される短いスケジューラジョブで、小さな設定可能な測定ツール一覧 (プロファイラ、ハードウェアカウンタユーティリティ、その類) の利用可能性を名前で確認し、マシンアーキテクチャを記録する。結果はチェックポイント生成物 (`platform_capabilities.json`) としてキャッシュされ、実行内で再プローブされない。このプローブは設計上ベストエフォートである: 区画未設定、スケジューラ欠落、あるいはタイムアウトを超えるキュー待ちは、いずれも何も書かない記録済みスキップへ降格する — プローブ失敗はすべての消費者を無制約のままにし、下流の何も変えない。

プローブされた事実はそのうえで — データとして、決して組み込みのハードウェア知識としてではなく — プラットフォーム盲目な計画が生じかねない 3 つの層で中継される。**アイディア**時にはそれらが研究トピックに織り込まれ、構想段階 (section 3.11) はプラットフォームが取れる測定だけを計画する: 実現可能性は下流で繕うのではなく源で固定される。**契約**時には、選ばれたアイディアのプランからメトリック契約を導く主張抽出器がそれらを検証済みの能力ノートとして受け取り、契約が不在のツールに依存する証拠を決して宣言しないようにする — 契約は一度鋳造されると凍結される (section 4.3) ため、二重に重要である。**ノード**時には、pin された義務メッセージ (section 3.6) が検証可能なほど不在のツールを列挙するプラットフォームノートを運び、実行中のエージェントが欠けたコマンドを苦労して発見するために ReAct ステップを費やさないようにする。**どの**ツールがプローブに値するかの知識は HPC スキルの設定可能な一覧に存在し、コアのオーケストレータとゲートはドメインに依存しないまま、測定されたものを中継するだけである。

3.10 研究メモリ

ノードは孤立して走るわけではない: 各ノードは祖先が確立したものを継承し、その代わりに子孫と論文執筆器のために記録を残す。ARI はこの記録を実行ごとの研究メモリに保持する。これは自由形式のログでは満たせない二つの要件に従う。第一に**ブランチ隔離**: ノードは自身の祖先のメモリのみを読め、兄弟や無関係なチェックポイントのものは決して読まないため、二つの並行する手がかりは互いを汚染できない — section 3.7 の系譜を定める祖先スコープ述語と同一である。第二に**基盤づけ可能性**: メモリは最終的に論文を支えるため、エントリは結果を単に言い直すのではなく結果の背後にある証拠を指し示せねばならない。そうして主張はそれを支える生成物まで遡れる。

証拠への索引としてのメモリ

ノードが何を生成したかの唯一の真実の源は、その構造化自己レポート — ノード完了時に書かれる `node_report.json` である (section 4.2 がそのフィールドを詳述する)。メモリエントリはこれらのフィールドを複製しない。短い検索可能テキスト、種別、元のノードレポートへのポインタ、そして結果が依拠する特定の生成物 — ログ、ソースファイル、出力ファイル — へのコンテンツハッシュ参照を持つ。したがってメモリエントリは証拠そのものではなく**証拠への索引**である: 引用する生成物はいつでも再ハッシュして、なお存在し一致することを確認でき、各参照を検証済み・欠落・不一致に分類する整合性検査となる。種別は固定語彙 —観測、実験結果、失敗事例、手順、内省、生成物要約、論文主張、再現性イベント — から取られ、読み手は必要な記録を正確に要求できる: あるブランチ沿いの失敗のみ、あるいは生成物に裏づけられた結果のみ、というように。

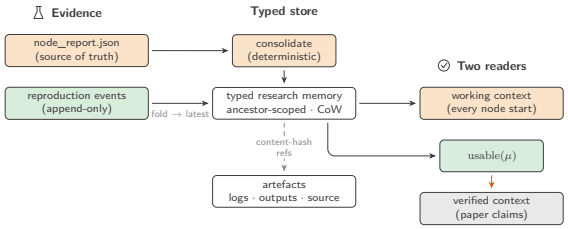

Figure 11: 検証可能な研究メモリ。各ノードの真実の源である `node_report.json` の決定的統合が、生成物 (ログ・出力・ソース) を内容ハッシュで**インデックス**する**型付き**格納庫を満たす。再現イベントは追記され最新状態へ畳み込まれ、許容述語 usable(μ) (eq. (6)) がどのレコードを検証済み文脈へ通すかを制御する。同じ格納庫が二つの読み手を養う: 次ノード開始時に注入される**作業文脈** (探索の継承経路) と、論文の定量的主張を基盤づける**検証済み文脈** (section 4.2) である。基盤のない内省は探索を導くことはあっても論文本文を導くことはない。

ノード終端での統合

ノードが完了すると、決定的な統合ステップがそのノードレポートを読み、型付きエントリを発する: 測定を生んだノードには、測定した生成物への来歴参照を伴う実験結果エントリを; 失敗したノードには失敗事例エントリを。このステップは言語モデルを呼ばないため、同じノードレポートは常に同じエントリへ統合される — 統合は記録された状態の純粋関数であり、チェックポイントの再実行はそれが生んだメモリを再現する。これはエージェントの動作ではなくループフックである: 探索エージェントが「メモリに保存せよ」と指示されることはなく、実際エージェントが自発的に想起ツールへ手を伸ばすことはない。ループが依存するメモリ読み出しはループ自身が決定的に行う — これが、非決定的な検索をクリティカルパスに置かずにメモリへ来歴を担わせる仕掛けである (section 7.1)。

二つの読み手: 作業文脈と検証済み文脈

格納庫は二つの消費者を養い (fig. 11)、両者の区別が設計の核心である。各ノードの**開始**時、ループは作業文脈を注入する: 実験の固定核 — 目標、対象メトリック、ハードウェア — と、祖先が記録した結論である。よって子は集約テキストダンプからではなく、系譜がすでに確立したものから、決定的かつ自身のブランチに限定して始まる。**論文**時には、パイプラインが最良ノードの系譜から検証済み文脈を構築する: 各結果の再現性イベントを最新状態へ畳み込み、再現された結

果が単に生成物に基盤づけられた結果より上位に、後者がさらに基盤のない内省より上位になるようエントリを順位づける。エントリ μ が定量的な論文主張として**許容**されるのは、それが生成物に基盤づけられ、かつ再現の試みに反証されていないときに限る:

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

基盤のない内省は作業文脈を通じて探索を導けるが、eq. (6) がそれを論文本文から締め出す: 執筆器の定量的主張は許容され再現を優先したエントリのみに依拠する。これは証拠アセンブラ (section 4.2) のメモリ側の対応物である; アセンブラが執筆器の読める生成物**バイト**を決めるのに対し、検証済み文脈はどの**結果**が主張になれるかを決める。

追記イベントとしての再現性

過去のエントリはバイト安定 — ノードは自身の識別子のもとでのみ書き、既存エントリは決して上書きされない — であるため、結果の再現性状態は後で再実行されても上書きされない。代わりに再実行は再現性イベントエントリを**追記**し、読み手は対象ごとのイベントを最新状態へ畳み込む。再現に失敗した結果は、それを有望と記録した歴史を書き換えることなく eq. (6) の主張集合から降格され —そして正直に限界として報告できる。

3.11 構想: ルートの播種

木のルートノードはコードから始まるわけではない; 何を調査するかを決めることから始まる。ルートのエージェントはアイディア段階を走らせる: 有界な文献調査に続いて多エージェントのアイディア生成を行い、選択されたアイディア (チェックポイント内の `idea.json`) へ収束する。そのプランが実行のメトリック契約 (section 4.3) を生む。調査はここで、構想時に行われるため、探索自体はオフラインに保てる (section 3.7): 調査は Semantic Scholar からライブの文献を取得し — キーワード検索を引用グラフの 2 ホップ走査で豊かにする — 続く討議は取得した抄録の上で働く。

構想は意図的に多エージェントである。単一エージェントの、アーカイブに条件づけられたブレインストーミングは、狭いアイディア分布へ崩落することが知られている — AI Scientist はそのアイディア生成が「異なる実行間で、さらにはモデル間でさえ、しばしば非常に似たアイディアをもたらす」と報告する [2] — 一方で VirSci [7] は科学者チーム (共同研究者選択、円卓討論、アイディア生成、新規性投票) をシミュレートし、多エージェントシステムが単一エージェントのエグゼクティブを、平均で、現代研究との整合性で +13.8%、潜在的影響で +44.1% 改善すると報告する。

ARI は VirSci を二つの忠実度でラップする。デフォルト経路はアイディアスキル内に VirSci の討論フローを再実装し、利用可能な場合は vendored の VirSci プロンプトテンプレートを直接用い、検索をライブ調査に再ベース化し、モデル呼び出しを ARI の LLM 層経由でルーティングする; 候補アイディアは VirSci 流の自己評価 (新規性、実現可能性、明瞭性) を伴い、新規性で重み付けされた和でランキングされる。オプトインの *VirSci-live* モードは代わりに、vendored の VirSci エンジン自体を未改変で駆動する (fig. 18):Semantic-Scholar 由来の凍結スナップショット — SPECTER2 検索インデックスと鮮度でランキングされた共著グラフを伴う論文コーパス — がチェックポイント下に実体化され、VirSci 自身の `select_coauthors` がそのグラフ上に科学者チームを形成し、その K ラウンドの `generate_idea` 討議が候補を生む。こうして ARI が構築の基盤とする多エージェント機構は、言い換えられたものではなく、現在の文献に対して実行される、公刊されたそのものである; スナップショットの凍結が討議基盤を再現可能に保ち、ライブ経路はいかなる失敗時にも再実装ループへ降格するため、これを有効化しても実行が退行することは決してない。

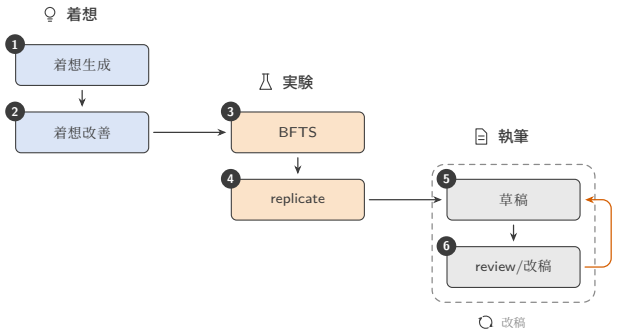

Figure 12: 論文生成のライフサイクルをフェーズでまとめた列として示す: 着想、実験、執筆 (ステップ 1–6)。破線の囲みはレビュー/改稿の対を表し、そのバックエッジがライフサイクル中で唯一の非単調遷移である (section 4.5)。

二つの実行レベルの事実が、両経路で討議に織り込まれる。プローブ (section 3.9) が検証したプラットフォーム制約が研究トピックに付加され、トピックを消費するすべてのプロンプトがプラットフォームの取れる測定だけを計画する; そして派生実行では先祖チェックポイントのアイディアプール (section 3.7) が読み取り専用で注入され、チームは系譜が既に提案したものを再導出する代わりに、それを洗練・拡張、あるいは明示的にピボットする。候補間のランキング付き選択は事後監査のため記録され (section 3.7)、実行の出発点をその後のどの拡張とも同じく追跡可能に保つ。

4 論文生成パイプライン

4.1 パイプライン概観

探索が停止すると、生き残った系譜が論文生成パイプライン——実行の後半 (fig. 12;fig. 3 はこれを実行レベルのステージ DAG の中に位置づける)——へ手渡される。執筆フェーズが系譜を草稿へ編纂し、レビューフェーズがその草稿をルブリックで採点する; 両者は改稿ループ (section 4.5) で結合される。パイプラインは 3 つの出力を永続化する:LaTeX 草稿、根拠付きの葉ごとレビュー、そして本実行の採点に用いたルブリックインスタンス (ルブリックは事前固定ではなく論文ごとに生成され得るため保存する)。

パイプラインの組織原理は、草稿中のあらゆる定量的言明が、探索が実際に生んだ何かへ辿れねばならない、という点にある。二つの機構がこの原理を担う。決定的な証拠組立器 (section 4.2) は、記録メタデータの純関数として、執筆器が依拠してよい成果物が**どれ**かを定

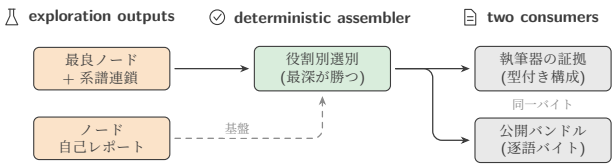

Figure 13: 証拠の組み立て。決定的な組立器は最良ノードの系譜を辿り、各ノードの構造化自己レポート (内容ハッシュ、成功自己評価、逐語のビルド/実行コマンド、捕捉ハードウェア) に基礎づけられて、**二つ**の消費者へバイト同一の内容を供給する **一つ**の役割別選定を行う: 執筆器の系譜証拠と、公開成果物バンドルである。各寄与ファイルは記録メタデータにより選ばれ、パス衝突時は最深の出現が勝ち、連鎖上の削除はそのファイルを除く。

める。主張 – 証拠契約 (section 4.3) は、論文がそれらについて**何**を述べねばならないかを定める: 着想時に宣言される反証可能な主張、各主張の証拠として数えられる測定名、そして論文が出荷される前に、報告されたすべての数値を記録済み結果から再導出する決定的ゲートである。残りの節——ルブリック形式 (section 4.4)、レビュー/改稿ループ (section 4.5)、PaperBench 複製経路 (sections 4.6 and 4.7)——は、この二つの上に構築される。

4.2 証拠の組み立て

探索と執筆の間には証拠組立器が座る (fig. 13)。これは系譜のノード単位成果物のうち執筆器が依拠してよいものを決め、それらを単一の科学向けレコードへ整形する。設計はこの継ぎ目を監査可能に保つ——ある主張がどのノードに辿れるかは、言語モデルの判断ではなく記録されたメタデータの関数である——ため、section 4.3 の契約と section 4.8 のガードレールが強制すべき具体物を持つ。

選別は役割別である: 一つのノードは三つの消費者へ異なる仕方で寄与し得て、単一の述語族が各々を判定する。ノードは、実測値を持つ (あるいは評価器がその実行を成功と認定した) とき**合成**集合へ入り; 親に対しソースファイルを追加・変更したとき**コード**集合へ入り; そのステップが成功したとき**物語**集合へ入る。行き止まりを探索しただけのノードはいずれにも入らない。各基準は記録メタデータ上の純粋述語であるため、同じ入力は常に同じ寄与集合を選ぶ。

ソースファイルは、最良ノード $n^\star$ の先祖連鎖 $\mathrm{lin}(n^\star)$ に沿って各寄与ノードが触れたファイルを、深さ順に重ね合わせて集める。子ノードは親の作業ツリーの複製内で実行されるため、一つの相対パスが複数の深さで現れ得る; 最も深い出現が勝ち、連鎖上の削除はそのファイルを除く——こうして復元集合は最良ノードの実作業ツリーに一致する。この選定は同じバイト列から**二つ**の消費者へ供給される: 執筆器の系譜証拠と、公開成果物バンドルである。ゆえに論文が記述する系譜コードは、公開されるコードに一致する。選定は連鎖限定である——最良系譜外のノードは、その実測が合成集合で報告されても公開されず、provenance のために記録される——加えて擬似コードの忠実性のための補助として、執筆器には上位スコアノードの生ソースも別途示されるが、それ自体は公開集合ではない。

組立器はコード・パラメータ・実測値を、要約ではなく各ノードの作業ツリーから**直接**読む。実験が型付き結果を宣言した場合、レコードは入力パラメータ (構成ノブ——問題サイズ、スレッド数、シード) を、実測出力 (スループット、精度、レイテンシ) および導出量 (効率、モデル予測の上限) から分離する。この分離は装飾ではない: 構成間で報告される軸ごとの極値は、決定的に、かつ実測キー**のみ**に対して縮約される、

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

ここで red はそのメトリックの宣言された方向に従う最大または最小、$\mathcal{E}$ は寄与構成の集合、$\mathcal{I}$ はあらゆる縮約から外される宣言済み入力キーの集合である。入力を外すことが、大きな入力規模——たとえば数百万の非零要素という問題サイズ——を主たる結果と読み違える事態を防ぐ。言語モデルは真にそれを要する唯一の仕事——逐語ソースとログを散文・擬似コードへ変換し、方法論を物語ること——に限定される。数値を供給することは決してない。

これらすべての基盤は、ノード完了時に書かれるノード単位の構造化自己レポート (チェックポイント内の `node_report.json`) である。これは内容ハッシュにより、当該ノードが親に対しどのファイルを追加・変更したか; ノード自身の成功自己評価と評価器が指摘した懸念; 逐語のビルド・実行コマンド; そして実測が動いたハードウェアを記録する。ハッシュはソース選択を再現可能にし、公開バンドルが記録コマンドから構築した再現スクリプトを携えることを可能にする; 捕捉されたハードウェアは、論文の環境節を「記録なし」のままにせず事実から書けるようにする。

Algorithm 2 証拠の組み立て (探索 → 執筆器入力)。

Input: ノード単位レポート付きの系譜ノード $\mathcal{N}$; メトリック方向写像
Output: 科学レコード $\mathcal{E}$: 型付き構成、軸ごとの極値、方法論物語

$n^\star \leftarrow \arg\max_{n \in \mathcal{N}} r(n)$ ▷ 同点: 検証済み、次に深い方
$L \leftarrow \mathrm{lin}(n^\star)$ ▷ 根から最良への先祖連鎖
$\mathcal{C} \leftarrow \{\, n \in L : \textsc{Contributes}(n) \,\}$ ▷ 役割述語 (コード)
$F \leftarrow \emptyset$ ▷ 相対パス ↦ (ノード, 深さ)
for all $n \in L$ を深さ順に **do**
  for all $n \in \mathcal{C}$ が追加/変更したパス p **do**
    if $p \notin F \vee \mathrm{depth}(n) \geq \mathrm{depth}(F[p])$ **then**
      $F[p] \leftarrow (n, \mathrm{depth}(n))$ ▷ 最深が勝つ
    end if
  end for
  for all n が削除したパス p **do**
    F から p を除く ▷ 削除が上書きで除去
  end for
end for
各 $p \in F$ のバイトをサイズ予算の下で逐語的に読む
for all 実測キー $m \notin \mathcal{I}$ **do**
  $\mathrm{best}(m) \leftarrow \mathrm{red}_c\, c[m]$ ▷ eq. (7); 決定的
end for
$\textsc{Narrate}(F)$ ▷ LLM: 逐語ソース/ログ → 散文、擬似コード
return $\mathcal{E}$

4.3 主張 – 証拠契約

パイプラインの中心はメトリック契約である: 最終的な論文が支えねばならない反証可能な主張の実行レベルの宣言であり、各主張は、その証拠として数えられる

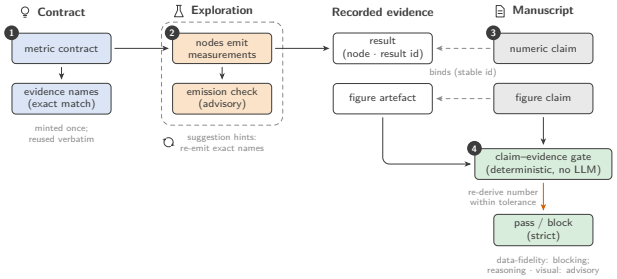

Figure 14: ARI による Story2Proposal の主張 – 証拠契約 [8] の execution-grounded な実現。(1) 実行レベルのメトリック契約が反証可能な主張と、その証拠として数えられる正確な測定名を宣言する; 主張を担う契約が一度永続化されると、それは逐語的に再利用される。(2) 探索ノードはそれらの名前の下で測定を発する; 発信時点の助言的チェックが未充足の主張をフラグし、再発信のための提案のみの名前ヒントを提供する——自動束縛は行われない。(3) 原稿の主張——数値の断言と図参照——は、安定なノード・結果・図の識別子により記録済み証拠へ束縛される。(4) 決定的な主張 – 証拠ゲート (言語モデルなし) が、報告された各数値をその証拠から許容誤差内で再導出する; データ忠実性の軸は厳格モードでブロックし、推論と視覚整合性の軸は助言的である。

正確な測定名を名指す (fig. 14)。主張は選択された着想に帰属するため、実装エージェントは都合の悪い主張を捨てられない; 契約は着想時にチェックポイントへ永続化され、実行全体を貫く。探索中、ノードは契約の名前の下で測定を発する; 同じ名前は未充足の主張へ向けて拡張を誘導し、section 3.8 の系譜連鎖の種となる。ゆえに、より早い測定から**計算**される証拠を、単一の系譜に沿って生成できる。執筆中、下記の決定的ゲートが主張と証拠を正確な名前で照合する。肯定的・否定的いずれの結果も主張を充足する; 契約が必須とするのは、それを判定するための証拠である。

この設計は Story2Proposal [8] に由来する。これは完成した「研究ストーリー」——物語とその実験的証拠——を、architect・writer・refiner・renderer エージェントの固定されたパイプラインに永続的な契約を通すことで原稿へ変換する。その共有契約は、原稿の視覚成果物を正準ラベルの下に登録し、それらをセクションレベルの義務へ束縛し、文書全体の検証規則を携える; 推論検証・データ忠実性・視覚整合性のための評価エージェントが中間成果物を観察し、生成中に契約を更新し、renderer が決定的な構造検証で締めくくる。その契約が保証するのは構造的・視覚的な整合性——一意なラベル、解決可能な相互参照、物語と視覚の整合——であり、著者らはその境界を明示する: このフレームワークは「欠落した科学的証拠を生成するのではなく構造的・視覚的義務を強制する」のであり、原稿は「あまり厳密でない推論を含みつつ構造的に妥当」であり得る [8]。

ARI は契約の発想と三つの評価軸を保ちつつ、契約を実行に再基礎づけする。宣言される対象は原稿の視覚構造ではない; それは実行の証拠的義務であり、いかなる実験が走る前にも宣言され、実行された実験が実際に発した測定へ束縛される。データ忠実性は対応して再実装される: 言語モデルの評価エージェントの代わりに、決定的ゲートが報告された各数値を記録済み証拠から再導出し、この軸は厳格モードでブロックする一方、推論と視覚整合性は助言的なままである。Story2Proposal は与えられた証拠がどう書き上げられるかを検証する;ARI はさらに証拠がどう存在するに至るかをも制御し、それがブロッキングチェックを健全にする。

具体的には、執筆器は報告される各数値の傍らに本文内アンカーを発する: その数値が支える主張、メトリック、導出式、そしてそれが計算されたオペランド構成を名指すコメント行の宣言である。決定的なリンカがアンカーを組み立て済みの科学レコードに突き合わせ、ゲートは次に、名指された式を用いて宣言された各数値を記録済み測定から再計算し、宣言された許容誤差 (既定では小さな相対許容誤差) 内で照合する。宣言は前向き専用である——数値が何を意味し得るかを逆探索で推測しない——ため、誤った宣言は黙って修復される代わりに不一致として失敗する。発信もまた生成時点で検査される: ノードが結果を発するとき、最終ゲートが後に走らせる存在チェックが直ちに鏡映され、ノードはまだ再発信できる; 警告は未充足の主張を名指し、必要な名前へ向けた保守的で提案のみの字句ヒントを携え得る。何も自動束縛されず、ゲート自身はヒントロジックを決して参照しない。

実際の実行の経験は、この継ぎ目の 4 点を強化した。いずれもプロンプトの手直しではなく構造的な規則である:

- **契約は一度だけ鋳造される (mint-once)。**主張を担う契約が一度永続化されると、以後のすべての呼び出しは、再生成する代わりに永続化済みの契約を逐語的に返す; 再導出要求は義務の無害な再読み込みになる。言語モデルの命名は参照として安定でない——二度尋ねれば同じ量に異なる名を与える——実際、ある実行では実行途中の再生成が証拠語彙を黙って改名し、兄弟ノードが既に発していた測定を完全一致ゲートから隠した: 証拠欠落と報告された主張は、実は以前の名前の下で測定済みであった。
- **解析は執筆器の現実に合わせる。**ゲートの数値検証は、科学的記数法をその二つの一般的な形式 (指数接尾辞と $m \times 10^k$ リテラル) の双方で単一の値として解析し、数式デリミタと間隔マクロを正規化して取り除くため、値 – 単位の組を抽出するとき数値とその単位が隣接する。実際に繰り返し現れる執筆器の宣言の癖——同じ導出に対する同義の式名、冗長なラベル接頭辞つきで書かれたオペランド参照、LaTeX エスケープを担うメトリック名、複数の言及で共有される一つのアンカー識別子 (言及ごとに曖昧性除去される)——は、不一致として報告される代わりに解析時に正規化される。理由は二重である: ゲートの判定は論文の表層構文ではなく数値を測るべきであること; そしてアンカー行は改稿中に編集され得ないため、解析時に行き詰まる宣言は下流で修復され得ないこと。
- **レビュー所見は欠けずに改稿器へ届く。**意味論レビュー——過大主張と裏付けのない因果的言明をフラグする補完的な言語モデルのパス——が上げるすべての警告は、切り詰めた部分集合ではなく、助言的な改稿エントリとして改稿ステップ (section 4.5) へ転送される; 転送されたエントリのない警告は決して改稿器に届かず、その件数も決して減らせない。改稿後の解消数は二つのレビュー

のパス間の生の差分であるため、新たな所見を持ち込む改稿は、丸められて消える代わりに負の差分――退行――として現れる。

- **ブロックは客観的な虚偽のために確保される。**ゲートの所見は二層で来る。ポリシーゲートされる所見――再計算に失敗した報告数値、解決できないオペランド、引用された証拠が欠落した数値の断言――は厳格ポリシーの下で最終チェックをブロックし、それ以外ではブロックせずに報告される。客観的な虚偽――破られた普遍的または宣言された不変条件、失敗または欠落した宣言済み正当性チェック、実測上限が要求される箇所のプレースホルダ定数、自身の生オペランドから再計算できないメトリック、実行のどこにも裏付け測定のない宣言済み主張――は**いかなる**ポリシーの下でも最終チェックをブロックする。各々が決定的に偽または不健全だからである。意味論レビューの主観的所見は設計上助言であり、判断の問題で公開をゲートすることなく改稿器を導く。

4.4 ルブリック形式

ルブリック R を、葉が組 $(c_i, w_i, \text{judge}_i)$ を保持する有限木としてモデル化する。各葉 i は、カテゴリ記述子 c_i、非負の重み w_i($\sum_i w_i = 1$ を満たす)、そして段階的スコアを返す**判官関数** $\text{judge}_i : P \to [0,1]$ を持つ。論文スコアは凸結合

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

で与えられる。この木構造ルブリックは PaperBench [9] と共有されており、そこでは二値の葉評点が重み付き平均として木を上昇する;ARI は vendoring された複製を通じて上流パッケージを再利用する (section 4.6) ことで、その評点・集約の意味論を上流に逐語的に追従させる。判官は二族が併存する: 定性的な葉のための LLM ベース判官と、二値または数値の検証器を持つ葉のための決定的判官 (例:「コストはドル絶対値で報告されているか?」) である。両者は同一の評価器インタフェースを満たすため、探索エンジン (section 3) と論文パイプラインは一つの継ぎ目を介して軸をスカラーへ縮約する。

ルブリックは二つの**モード**のいずれかで生成される。*agent-benchmark* モードでは木を論文の科学的貢献で分解し、各葉は候補 submission の実行出力が論文を再現するかを問う――元来の PaperBench 枠組みである。*paper-audit* モードでは木の最上位の子が固定の再現性軸集合となり、各葉は論文本文そのものの記述完全性を採点する――HPC 再現性監査を意図した枠組みの反転で、問いは「エージェントは論文を再現できるか?」ではなく「論文は再現に十分な記述をしているか?」となる。venue 知識 (どの軸を、どう問うか) はエンジンに焼き込まず venue ごとのテンプレートで与えるため、ARI コアはドメイン非依存 (P4) のまま保たれ、新規 venue はコードではなくデータの変更で済む。

paper-audit スコアを退化領域――空の submission が submission 主語の葉をすべて「No」に追いやる領域――から引き上げるには、葉の問いを (欠落した)submission ではなく**論文**を主語に問い直し、判官に本文と並べて論文の図を与え (視覚対応モデルが図を活用できるように)、任意の Artifact Description / Artifact Evaluation 補遺を追加入力として許容する必要があった。これらの調整はすべて ARI の wrapper 層に留まり、vendoring された判官は無改修であるため、上流バージョン間でスコアが比較可能に保たれる。論文自身の報告値から再現ログを合成することは意図的に**行わない**: それは executed-submission 段階を短絡させ、leaderboard 実行と比較不能な形でスコアを嵩上げしてしまう。

4.5 レビュー/改稿ループ

改稿サイクルはレビューのフィードバック $J(P_t)$ の下で P_t を P_{t+1} へ反復的に書き換える:

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

サイクルはレビュアが草稿を受理した時――収束基準 $\text{score}(P_{t+1}) - \text{score}(P_t) < \epsilon$ としてモデル化――、またはセクションごとの小さな改稿上限に達した時に停止する。どちらの分岐がより頻繁に発火するかは、section 5 がパネル規模で答えるべき経験的問いであり、本章では定量主張を行わない。

フィードバック $J(P_t)$ は、その源を区別したまま統合された改稿ストリームである: 独立した venue レビュー (section 4.4 のルブリックインスタンス)、証拠に基礎づけられた意味論レビューを丸ごと転送したもの (section 4.3)、そしてゲート所見から導出された機械的な改稿エントリである。統合は構造的である――言語モデルがストリームを合成しない――ため、改稿器は各所見をその元の語で見て、venue レビューの独立性が証拠に基礎づけられたチェックと並んで保たれる。

改稿ステップは軸ごとに意図的に非単調である: 明瞭性の問題を直すと数値軸がわずかに悪化し得る。これが fig. 12 のバックエッジの源であり、収束基準を軸ごとではなく集約スコアに対して定義する理由でもある――ただし軸ごとの下限 (section 4.8) を併せて設け、集約値が単一軸の崩壊を覆い隠さないようにしている。

4.6 PaperBench 統合

複製経路は PaperBench [9] の再実装では**ない**: 上流パッケージを無改修で vendoring することで ARI はそのタスク・ルブリック意味論を逐語的に追従し、ARI はその周りの薄い wrapper のみを提供する。wrapper はプロトコル準拠のループを、共通語彙を持つ 3 つの合成可能なステージとして公開する (fig. 15):

1. *rollout* ―― 複製エージェントが論文から submission (コード、および任意で再現スクリプト) へ向けて作業する;
2. *reproduce* ―― submission を隔離コンテナ内で、ローカルマシン上またはクラスタへディスパッチして実行する;
3. *grade* ―― vendoring された判官が submission をルブリック木に対して採点する。

二つの設計選択がこれをベンチマークに忠実に保つ。第一に、採点は vendoring された判官に委ね、ARI のアダプタはその出力を整形し反復実行を単一スコアへ

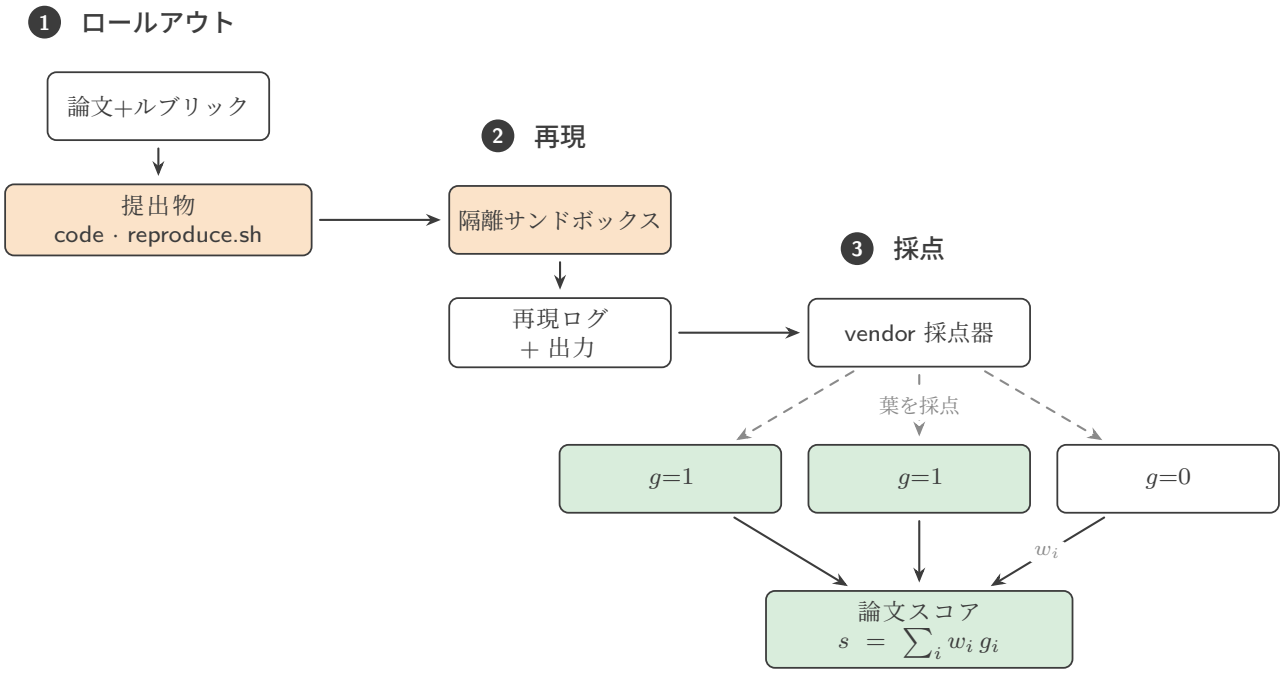

Figure 15: PaperBench 複製経路を 3 つの合成可能なステージとして示す。エージェントの *rollout* は論文とそのルブリックを submission(コードと再現スクリプト) へ変換し、*reproduce* はその submission を隔離サンドボックス内で実行して再現ログとその出力を取得し、*grade* は submission を**重み付きルブリック木**に対して採点する。vendoring された判官が各葉に評点 $g \in \{0,1\}$ を与え、論文スコアは重みで集約したロールアップ $s = \sum_i w_i g_i$ となる (eq. (8))。

集約するだけなので、PaperBench の更新は葉採点の結合方法を変えない vendor swap となる——同じエンベロープが eq. (8) に流れ、二値葉では $\text{judge}_i \in \{0,1\}$ となる。第二に、wrapper は *fail loud* である: ベンチマーク公正な実行が提供できない場合 (コンテナランタイム無し、スケジューラ無し、GPU リソース登録無し)、弱いホストへ黙って降格せず例外を上げる。黙った降格は結果スコアを leaderboard エントリと比較不能にするためである。旧来の silent-fallback 挙動は明示的 opt-in でのみ利用できる。

第二の関心事は、複製エージェントを**ホストが実際に提供するもの**について誠実に保つことである。上流プロンプトに委ねると、エージェントはホストに存在しないバイナリを呼ぶ再現スクリプトを scaffold しがちで、論文の実言語に関わらず Python を既定にしがちである。ARI は両者を、scaffold 前に環境を probe するようエージェントを誘導し、利用可能なツールチェイン・GPU・ネットワーク到達性のライブ probe から生成した per-host のノートブロックを注入することで打ち消す——エージェントが読むものとホストが実際にできることを一致させる。

4.7 ドメイン幅: 実行プロファイル契約

PaperBench 上流は単一 GPU の単一コンテナを仮定する。方法論が本質的に並列な論文——多ランク MPI カーネル、多ノード学習、クラスタ固有のモジュール環境——を覆うため、ARI はルブリックを任意の実行プロファイル (fig. 16) で拡張する。プロファイルはドメイン非依存である: 論文の科学分野ではなく**並列実行の形** (単一 CPU、単一/多 GPU、MPI、MPI+GPU) を、論文のスケール包絡やスケジューラヒントと共に名指す。論文が単一マシンならプロファイルは省略され、パイプラインは元の単一ノード呼び出しに退化する: 契約は加法的であって破壊的ではない。

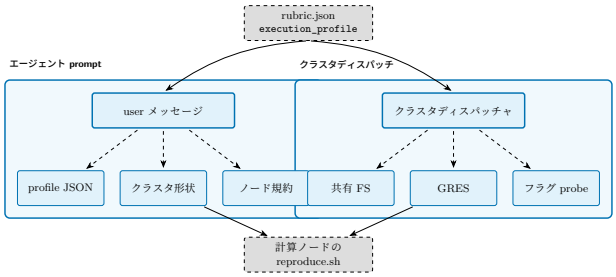

Figure 16: 実行プロファイル契約は単一のルブリック成果物から二つの消費者へ流れる: 複製エージェントのプロンプト (左) とクラスタディスパッチャ (右)。ランタイム probe がディスパッチする資源要求をローカルスケジューラに適応させるため、一つのルブリックが異種クラスタ上で無改変のまま忠実に動く。

プロファイルは二つの消費者に供給される (fig. 16)。エージェントのプロンプトには簡潔な「計算ノード実行規約」ブロック——共有ファイルシステムの規律、素の MPI ランチャより スケジューラのランチャを優先、環境活性化、wall-clock ラッピング——として付加され、クラスタディスパッチ (ノード・GPU・メモリ・配置要求) を媒介する。クラスタが受理するスケジューラフラグは一致しないため、ディスパッチャは実行時にローカルスケジューラを probe し非対応フラグを落とす。こうして同じルブリックが、多 GPU クラスタでも、GPU 登録の無いサンドボックスパーティションでも、単一ワークステーションでも忠実にディスパッチする。逆向きの保証も回帰ガードで強制する: 無関係なクラスタ割り当て内で動く ARI は、非 HPC 論文のプロンプトにクラスタ/MPI 固有の文言を決して漏らしてはならない。変わるのはこの wrapper 層だ

けで、vendoring されたベンチマークと ARI コアは拡張を通じて不変である。

4.8 故障様式とガードレール

4 つの故障様式は名前を与えるに足るほど頻出する。いずれも仮想ではない: 最も近い既刊の先行研究は、実結果のみを用いよというプロンプトレベルの指示にもかかわらず、実験ハードウェアを誤報告し結果テーブル全体を幻覚する生成草稿を記録している [2]。ARI の立場は、各故障様式が助言的でなく**構造的**な緩和策を要するというものである:

- **LLM のみの裏付け**: 論文が、系譜のどのノードも裏付けない主張を述べる。緩和策は執筆器を事前選択した系譜成果物集合に制約することで、数値的・因果的主張がそれを生んだノードまで辿れるようにする; 草稿は生成されていない証拠を引用できない。対になる制約はメモリ側にある: 検証済み文脈 (section 3.10) は、生成物に基盤づけられ再現の試みに失敗していない結果のみを定量的主張として許容する (eq. (6)) ため、再現された結果が優先され、裏付けのない内省は見出しの数値になれない。
- **引用ハルシネーション**: 論文が存在しない文献を引用する。緩和策は構造的である——執筆器は引用キーを発明できず、各 bib エントリは採用前に外部インデックスに対して検証される (section 6)。
- **改稿ドリフト**: 集約スコアは上昇するが論文が一貫性を失う。緩和策は軸ごとの下限である: いずれかの軸が閾値を下回れば改稿は中止され、ある軸での向上が別の軸の崩壊を覆い隠せない。
- **数値ドリフト**: 報告された量が、それが要約する記録済み結果と一致しない。各数値主張はそれが依拠するノードと測定値を名指す (section 4.3) ため、決定的な主張 – 証拠ゲートが各々を記録済み結果から再導出し、許容誤差内で照合する (fig. 14); 再計算値と一致しない値はフラグされ、section 4.3 の二層ブロッキングポリシーが、その所見が最終チェックで草稿を却下するか修復のために報告されるかを決める。

第一のガードは context も制限する: 執筆器には選択された系譜成果物と上位スコアノードのソースがサイズ予算の下で渡されるため、長時間実行でも草稿はモデルの context 窓内に収まる。

5 予備的観察

本章では、稼働中のシステムが v0.9.0 の段階で確立していることを報告する。証拠は三種類からなる: 公開された論文がすべての検証ゲートを通過したエンドツーエンドの生成実行 (section 5.1); 構成から成り立ち、かつ運用において確認される性質 (section 5.2); そして当時の v0.8.0 パイプライン上で実施した、より早い時期の再現演習 (section 5.3) である。これらはすべて予備的である: エンドツーエンドの観察は、制御されたベンチマークではなく単一の実行であり、それらが要請するパネル規模の研究は先送りされている (section 5.4)。本レポートの他所にある定量的な図は説明用であり、文書ビルドが再現可能となるよう固定シードのサンプルデータから生成している。

5.1 ケーススタディ: ゲート検証済みのサンプル論文

エンドツーエンドの結果が一つ実証され、検分可能である: 公開されたサンプル論文——実際の一回の実行が自律的に生成した 8 ページの研究——である。同論文は CSR(compressed sparse row) 形式の疎行列と密行列の積 (SpMM) を研究し、実行自身の測定からフィットされた性能モデル (論文の言う「loopline」モデル) の導きの下で、右辺 (RHS) 幅にわたるストア方策を選択する。この実行をエンドツーエンドで追跡する (fig. 17(section C に全文を収録)) のは、それが本レポートの記述する検証機構の存在証明だからである。

主張の構造。 実行のメトリック契約——アイデア時に一度鋳造され、実行の残りの期間にわたって逐語的に返される (section 4.3)——は、反証可能な主張を 12 件宣言した。うち 9 件は、単一のノードが直接測定する**独立** *probe* である: 密な参照実装に対する正しさ、ストア方策の比較、不規則 gather と局所性破壊の probe、ページフットプリント (translation lookaside buffer, TLB) の負荷 probe、計算レートの probe、帯域幅マイクロベンチマークである。残る 3 件は**計算された証拠の連鎖**を成す——性能モデルのパラメータをフィットし、そのフィットを held-out の probe 測定で交差検証し、次いでフィット済みモデルから構成を選択する——もので、各主張は新たな何かを測定するのではなく、前段の出力を消費する。

系譜の閉包。 この連鎖こそ、本実行が実証のために存在する事例である。計算された主張は、入力を持たないノードによっては証拠づけられない。そして先行する三つの実行がその失敗モードを裏づけていた: 兄弟 probe からモデルをフィットせよと指示されたノードは、代わりに単一の probe を再実行し、連鎖の主張は未充足のまま残った。ここでは連鎖が閉じた。子ノードが親 probe ノードの作業ディレクトリを継承し、継承した測定からフィット・交差検証・選択を計算したからである——系譜連鎖 (section 3.8) が「すでに手元にあるデータから計算する」ことを、局所的に利用可能な手として成立させた。そうして最終の主張 – 証拠ゲートは 12 件すべての主張が基盤づけられていると判定し、エラー 0 件を報告したので、論文は最終化された。

ゲートが検証したもの、そして論文がヘッジしたもの。 この結果の価値は**検証済み**という性質にあり、いかなる見出しの数値にもない: 公開された論文が主張するすべての定量的主張は記録済みの生成物に束縛され、決定的ゲートによって再導出される。そして論文は、言明が生成物の基盤を超えそうな箇所ではどこでも明示的にヘッジしている (table 1)。論文は、その強い定量的主張を正しさの指標——独立な密の参照実装に対する単精度浮動小数点 (FP32) の最大絶対誤差 6.79×10^{-6}——に限り、ストア方策のクロスオーバー、帯域幅の上限、モデルの数値予測は、基盤づけられた結果では

基盤づけ済みとして報告	正しさ: 独立な密の参照実装に対する FP32 最大絶対誤差 6.79×10^{-6}; 宣言された 12 件すべての主張が発行済みの証拠を伴う; 計算された証拠の連鎖が実行された。
明示的にヘッジ	ストア方策のクロスオーバー値、倍精度誤差、測定された帯域幅の上限、性能モデルの数値予測——方法論として述べられ、本改稿では生成物に基盤づけられていないと明記される。

Table 1: 公開されたサンプル論文は、ゲートに基盤づけられた数値のみを主張し、それ以外は自身の本文中でヘッジする——決定的ゲートとレビュー – 改稿ループが、論文を実行の支持できる範囲にスコープ限定する。

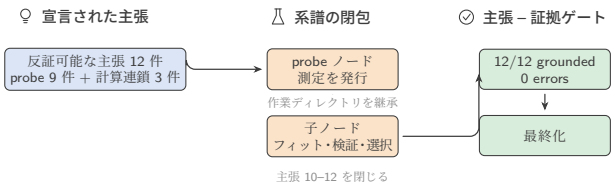

Figure 17: 主張 – 証拠契約の実例としてのケーススタディ実行。宣言された 12 件の主張は、9 件の独立 probe と 3 件の計算された連鎖に分かれる; 連鎖は系譜によって閉じる——子ノードが親 probe ノードの作業ディレクトリを継承し、継承した測定からフィット・交差検証・選択を計算する——そののち決定的ゲートが全主張を基盤づけられていると判定し、論文が最終化される。

なく方法論として提示する。この誠実なスコープ限定はゲートの効果であり、意味論レビューによって補強される: そのレビューは草稿中の過大主張 5 件を指摘し、改稿パスは 5 件すべてを解消した。解消件数は生の前後差分として記録される (section 4.3) ため、新たな過大主張を導入してしまった改稿は、回帰として表面化したであろう。

我々はこの結果を狭く読む。この性質を述べる価値があるのは、より早い時期の自律パイプラインがそれを欠いているからである: 最も近い既発表の先行研究では、ハルシネーション制御はプロンプトレベルであり、完成した草稿の自動レビューはブロッキングではなく助言的であり、生成された論文は実験の詳細を幻覚すると観察され、著者らは報告された結果の自動検証を今後の課題として挙げている [2]。単一の検証済み実行は、契約・系譜連鎖・ゲート・レビュー – 改稿ループがエンドツーエンドで合成されることを示すが、分布を確立するものではない——それこそが、下で先送りする制御された研究である。

5.2 構成から成り立つ性質

さらに三つの性質が構成から成り立ち、かつ運用において確認される。コスト計上は厳密である: 各モデル呼び出しにドルコストが付与され、実行ごとにチェックポイントの結果レコードへ集計される。探索の終了は予算のみによるものではない: 停滞検出器 (section 3.5) が内容に基づく退出を提供し、ステップ予算とコスト予算 ($T_{\max}$ / $C_{\max}$) が尽きる前に発火することが観察されている。そして実行プロファイル経路は仕様どおりに振る舞う (section 4.7): ルブリック生成は、HPC(high-performance computing) 論文に対しては論文の報告するスケール包絡を伴う多ランクのメッセージパッシングインターフェース (MPI) 実行プロファイルを発行し、単一マシン論文に対してはそれを省略する。一方、マルチフラグのクラスタディスパッチは、ランタイム probe が当該スケジューラの非対応フラグを落とした後、ローカルスケジューラに受理される。

5.3 より早い時期の再現演習 (v0.8.0)

システム最初の運用上のエンドツーエンド演習は v0.9.0 に先立つ: 当時の v0.8.0 パイプラインを、既発表の非可逆圧縮の対象——GPU 上の誤差限界つき科学データ圧縮器——に対して、section 4.6 の再現経路を通じて実行した。この単一対象で達成された再現の度合いは、部分的ではあるが実質を伴うものであった。エージェントは、合成データで代替するのではなく論文が引用するシミュレーションデータセットの実スライスを取得し; ネイティブの CUDA 補間カーネルをコンパイルして GPU 上で実行し; レベルごとの自動調律を伴う Lorenzo および多段補間予測器を実装し; 誤差限界の範囲を掃引して、およそ 50 から 90 dB のピーク信号対雑音比 (PSNR) の再構成品質を、対応する圧縮率とともに復元した。きめ細かなルブリックに照らせば、この実行が通過したのは重み付き葉のおよそ十分の一の水準であった。その理由は、パイプラインではなくエージェントの振る舞いに根ざす二つにある: 採点された指標は CPU 予測器から得られたものであり——GPU カーネルはビルドされ実行されたものの、エージェントはその予測品質を報告結果から省いた——またエージェントは時間予算のわずかな割合を消費した後に自発的に終了し、残るデータセットと独自実装の可逆ステージを未着手のまま残した。両方の振る舞いは、複製エージェント一般について文書化された失敗モードと一致する。すなわち、ほとんどのモデルは完了を主張して実行を早期に終え、コードを書くことについては、論文の結果を実行して一致させることよりもはるかに高いスコアを得る [9]。本演習は section 5.1 の検証機構に先立つものであり、歴史的なベースラインとして手を加えずに報告する。

5.4 限界

評価は依然として予備的である。上記のいずれの結果も制御されたベンチマークではない: 生成実行は単一の軌跡であり、再現演習は単一の対象論文を扱う。制御された評価——凍結チェックポイント上での中央値の実行コスト、シード別の探索曲線、カテゴリ別のルブリック分布、ならびに対照的な対象論文のパネルにわたるフルロールアウトの複製スコア——は、今後の課題として残す。同じ研究は、本レポートが未解決のまま残す経験的な問い——たとえば、どの終了原因 (section 3.5) が実際に最も頻繁に発火するか——にも決着をつけるべきである。

6 関連研究

ARI を 4 つの先行研究の流れに対して位置付ける。

6.1 コードのための木探索

AIDE [5] は機械学習工学をコード最適化として捉える——解はステートレスな目的関数で採点されるスクリプトであり——、試行錯誤を、draft・debug・improve の辺で結ばれたスクリプトの木上の探索として形式化する。MLE-bench [6]——実際のメダル閾値に対して採点される 75 件のオフライン Kaggle コンペティション——において、この scaffold は支配的である: 同一モデルが、2 つの汎用 scaffold の下では 0.8% と 4.4% のコンペでメダルを獲得するのに対し、AIDE の下では 8.7% のコンペでメダルを獲得する。したがって scaffold の設計が、素のモデル能力を上回る。ARI は解の木の骨格を継承するが、AIDE のハードコードされた貪欲方策——AIDE の著者自身が局所最適のリスクとして指摘している——を、LLM 駆動の選択器 (section 3.3) で置き換える。この選択器は `has_real_data`、metrics 全体、深さ、ソフトな `diversity_bonus` を一括で重み付けし、さらに section 3.7 の実行間系譜を追加する。また、両研究が目的関数をタスクとともに受け取る——MLE-bench の著者が認めるように、実際の研究は「明確な問題設定さえ持たないかもしれない」——のに対し、ARI は自らそれを鋳造する:metric contract(section 4.3) が、何をエビデンスとして数えるかを探索に先立って宣言する。

AlphaCode [10] は generate-and-filter の規範的な成果である:1 問あたり最大で数百万のプログラムをサンプリングし、タスク付属の example test で濾過し、最大で 10 件の提出にクラスタリングして、10 件の模擬 Codeforces コンテストにわたり平均で参加者の上位 54.3% に到達する。このレシピは、安価でタスクが供給する検証器を前提とする; ARI の候補は、与えられた検証器なしにクラスタ資源上で実行される実験であるため、広さは小さなフロンティアの best-first な舵取りに、安価な濾過は実行自身のエビデンスに対する決定論的チェックに、それぞれ取って代わられる。

6.2 自律研究エージェント

AI Scientist ファミリ [2, 3] は、同様のループ (idea → experiment → paper) を、報告では 1 論文あたり $15 未満のコストでエンドツーエンドに閉じ、実際の学会投稿に対して人間に近い均衡精度を持つ自動査読器を備える。しかしその査読器は**助言的**である——完成した PDF をランク付けするものであり、ハルシネーション制御はプロンプトレベルにとどまる——、そして著者らは、ハルシネートされたハードウェア、捏造された ablation 表、改善として語られた負の結果を列挙したうえで、結果の自動検証を将来課題として挙げている。ARI はその対偶である: 実験は idea あたり 1 本の線形軌跡ではなく BFTS の下で枝分かれし、metric は最初の鋳造時に凍結される実行レベルの artefact であり (section 4.3)、そして検証はブロックする——何を公刊してよいかを決めるのは、査読器のスコアではなく決定論的な claim–evidence ゲートである (section 4.8)。

Agent Laboratory [4] はエージェントを駆動者ではなく協働者として扱う: 人間が与える研究着想を必要とし、計画への遵守度を採点する LLM 報酬モデルの下で平坦な 2 プログラムのプールを変異させ、論文を模擬査読器でゲートする。その最も示唆的な結果は負のものである: それらの査読器は、10 点満点で人間の

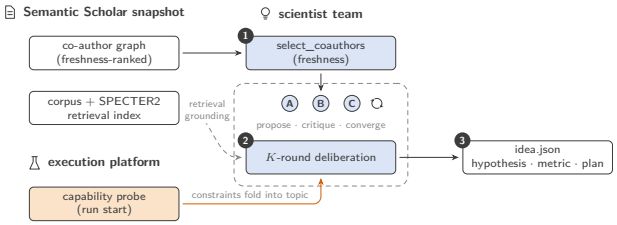

Figure 18: ARI が駆動する VirSci 流の着想生成。`select_coauthors` が、凍結された Semantic Scholar スナップショット (SPECTER2 検索インデックスを備えたコーパスと、鮮度順の共著者グラフ) から不均質な科学者チームを編成する; チームはスナップショットに対する検索で基盤づけられた K ラウンドの熟議を行い、実行開始時の能力プローブが、プラットフォームの計測された制約を熟議トピックへ畳み込むため、収束した `idea.json` は、プラットフォームが実際に計測できる測定だけを約束する。

査読者を 2.3 点過大評価する——これは、全体論的な LLM-judge によるゲートが不十分である理由の、最も明確な定量化である。したがって ARI の意味論的レビューは舵取りはするが決してブロックせず、ブロックは決定論的ゲートに留保され、すべてのノードは過渡的なプールではなくチェックポイントスコープの木の中に永続する。

VirSci [7] は、これらのシステムが最も希薄に扱う段階をカバーする: 研究コミュニティのデジタルツインに基盤づけられた科学者チームであり、その多エージェントの熟議は、現代研究との整合性と現代研究への潜在的インパクトにおいて、単一エージェントのベースラインをそれぞれ 13.8% と 44.1% 改善する。それは embedding 近接性の新規性プロキシで採点される abstract で止まり、その著者らはより厳密な下流検証を求めている。ARI はこの分業の両半分を引き受ける:VirSci の元の熟議エンジンを直接かつ無改変で駆動し (fig. 18)——鮮度に基づく `select_coauthors` のチーム編成と多エージェントの `generate_idea` ループ——、それをライブな Semantic Scholar スナップショット上で動かすため、依拠する機構は、現在の文献に対して実行された、公刊済みのそのものとなる; そして繰り延べられた検証を下流で供給し、そこでは claim–evidence ゲートの下の実行済みエビデンスが、プロキシの新規性スコアを置き換える。

6.3 論文評価

PaperBench [9] は、rubric-as-tree 形式の先駆者である: その 20 件の対象論文の各々が、著者が共同開発したルブリック——合計で 8,316 の採点可能な葉——を備え、その葉ごとの採点は pass/fail で、重み付き平均によって集約される。また、ハードコードされた結果が複製を装えないように、クリーンスタートの再現フェーズを設ける。ARI はこのベンチマークを無改変でベンダリングし (section 4.6)、同じ葉ごとの採点と集約重み (eq. (8)) を採用したうえで、改稿ガードとして使う軸ごとの下限を追加する (section 4.8)。その実行ギャップ——最強のエージェントがコード開発で 43.3% を得る一方、実行では 4.5%、結果の一致ではゼロである——は、事後に採点するのではなく検証を生成の内側へ移す動機づけとなる; そして手作りのルブリック (原著者とともに 1 論文あたり数週間) は、自動ルブリック構築

を将来課題として残す。ARI が選ぶのはこの枝である: ルブリックは論文ごとに生成されるため (section 4.4)、手作りルブリックの再現と leaderboard 比較可能なスコアは、設計上スコープ外である。

Story2Proposal [8] は、永続的な共有コントラクト——セクション構造、視覚的 artifact のレジストリ、文書全体の検証規則——を通じて構造化された原稿生成を統治し、writer・refiner・renderer エージェントが、推論検証・データ忠実性・視覚的整合性で採点される generate–evaluate–adapt ループの下で協調する。その利得は 4 つの LLM バックボーンにわたって保持されるが、エビデンスは与件として受け取られ、その failure-mode 分析は、原稿が「あまり厳密でない推論を含みながら構造的には妥当でありうる」ことを認めている: コントラクトは構造を縛るが、証拠としての妥当性は縛らない。ARI はこの原理とその評価軸を、実行に基盤づけられたパイプラインへと持ち込む (section 4.2): コントラクトは、安定なノード・結果・図表の識別子を参照する claim と numeric assertion で実験記録を拡張する; データ忠実性は、各報告値を記録済み結果から再導出する決定論的ゲートとなる; 推論と視覚的整合性は、独立した原稿査読に並ぶ非ブロッキングの evidence-grounded review となる。実験自体を実行することで、ARI は原稿レベルのコントラクトを execution-grounded な provenance へと拡張する。

6.4 基礎

ノードごとのエージェントループ (section 3.6) は Reason-and-Act(ReAct) パターン [1] に従う。その正確な発見はトレードオフである: 思考を基盤づけられた行動と交互に行うことは、ハルシネーションを急峻に削減する一方で、推論誤りの率を上昇させる。改稿サイクルは Self-Refine [11] と Reflexion [12] に由来し、それぞれ ARI が真剣に受け止める注意点を伴う:Self-Refine の利得はモデルが自らの誤りを検出できない場面では消失し、Reflexion の ablation は、改善を駆動するのが記憶だけではなく基盤づけられた評価であることを示す。したがって ARI は、事実に関する事柄についてモデルが自分自身を採点することを決して許さない: 改稿は、writer の外側で計算されるルブリックスコアとゲート判定によって舵取りされ、結果が定量的な claim となるのは、それが artefact に基盤づけられたときに限られる (section 3.10)。**思考**におけるスクラッチパッドは chain-of-thought プロンプティング [13] に由来し、その利得はモデル規模に伴って創発的であると報告される——その著者らはモデルが推論するとは決して主張していない。これらのいずれも、単一の軌跡を超える状態を提案していない; 実行を再現可能にするチェックポイントスコープの状態こそ、ARI の貢献である。

初期の草稿はさらに別の研究エージェントベンチマークを引用していたが、section 4.8 の文献検証規則を通る候補ソースが存在しないことが判明した時点で、その参照を取り除いた。したがってここでの網羅性は、Semantic Scholar インデックスに対して検証されたエントリに限定されている。

7 議論と限界

7.1 決定性 vs 新規性

P2、決定性原則は本システムの拘束的制約である。それは研究メモリ (section 3.10) が埋め込みモデルを使ってよいか否かを制約するのではなく、結果として生じる非決定性が**どこで**作用しうるかを制約する。メモリは生成的な言語モデル呼び出しを一切行わず、その埋め込み順位づけによるアーカイブ検索 — 遠隔サービスに依存し、ゆえにバイト単位では再現できない — は再現経路から外れた任意の想起補助として留まり、ループが決してそれを引かず、それ単独では論文主張を基盤づけられない。ループが実際に依存するメモリ読み出し — 各ノードで注入される作業文脈、論文を基盤づける検証済み文脈 — は、決定的でかつ祖先スコープの、ノードごとのレポートの畳み込みである。再現性の勝利のすべてに、こうした非決定性への扉を閉じるという代償が伴う; 下流ユーザとの契約は実行ではなくチェックポイントであるため、我々はこの代償を受け入れる。

7.2 構造的継ぎ目における指示追従

v0.9.0 の繰り返し現れる教訓は、指示追従は構造的継ぎ目 — 解析すべきインデックス、再利用すべき名前、宣言すべき数値 — において劣化するということ、そしてその恒久的な対処はプロンプティングではなくパーサ側での吸収であるということである。解析不能な選択インデックスは決定的な順位づけにフォールバックする (section 3.3); 二度問われたモデルは同じ量を異なる名前で呼ぶため、metric contract は一度だけ鋳造され逐語的に再生される; ライタの繰り返し現れる数値宣言の癖は解析時に正規化される (section 4.3)。吸収された癖のカタログは実際の実行から育てられた反応的なものであり、ゆえに予期されなかった変種はまずゲートの不一致として表面化する — そしてそれは保守的に行われる: それはフラグ付けされ、未解決の不一致は最終チェックをブロックし、何も黙って受理されることはない。

同じ規律が、何がブロックしうるかを定める。ゲートはすべての定量的主張が記録された成果物と一致することを認証する一方、意味的レビューの主観的所見はリファイナ (section 4.5) を方向づけるものの公表を決してゲートしない — [2] の完全に事後的なレビューとは異なり — なぜならブロッキング経路上の主観的判定は監査不能な拒否権になってしまうからである。したがって検証済みの論文は依然として退屈なものでありうる。

7.3 スケーリング限界

BFTS 実行ループの基盤にある単一マシン仮定は最大のスケーリング制約である: 単一の `ThreadPoolExecutor` (section 3.2) が唯一のワーカープールであり、四ワーカーに上限づけられている。マルチホスト拡張には、明示的系譜規律 (P4;section 2.5) を先祖歩行だけでなくマージにも拡張する必要がある — これは計算された証拠を制約するのと同じギャップであり、その連鎖 (section 3.8) は今日、証拠依存グラフ上の明示的スケジューラによってではなく、単一の系譜に沿った名前

ヒントによってスケジュールされている。別の制約はLLM 採点者であり、経験的には探索器と並ぶ支配的な費用項目である; その出力を (論文ハッシュ、ルブリックハッシュ) でキャッシュすることは助けになるだろうが、まだ出荷されていない。

7.4 展望

最も広いギャップは評価の幅である:section 5 は一つの再現演習 (当時の最新の v0.8.0 パイプライン上で実行された) と一つの検証済み生成論文に依拠している — これは機構が組み合わさることの証拠であって、分布ではない。今後の課題は三つある: 凍結チェックポイントと対照的な対象論文のパネルにわたる統制された研究、すなわち [6, 9] のプロトコル精神に倣ったもの; 単一スカラーの集約に代わる、繰り返しのロールアウトと採点者パスにわたる分散を考慮した集約、判定者自体がノイジーであることを踏まえたもの; そしてより広い領域 — 契約が名前づけられる証拠の種類をスカラー測定の枠を超えて広げること、ちょうど実行プロファイル契約 (section 4.7) が ARI のホストできる計算を広げたように、主要な証拠が単一の指標ではない分野へと向かうことである。

A　記号表

本付録は本文で使われるすべての記号とその意味をまとめる。

記号	意味
$\mathcal{N}$	探索木のノード集合
$\mathcal{T}$	探索木 $(\mathcal{N}, E)$
$n \in \mathcal{N}$	個別のノード
$\mathrm{ch}(n)$	n の子集合
$\mathrm{pa}(n)$	n の親
$\mathrm{depth}(n)$	n の深さ
$s(n)$	ノード状態 $\in \{\mathrm{open}, \mathrm{eval}, \mathrm{done}, \mathrm{failed}\}$
$r(n)$	n のスカラー報酬
$\mathbf{e}(n)$	軸ごとの評価ベクトル
ϕ	軸 $\to$ スカラー集約器
$\mathrm{div}(n)$	多様性ボーナス (eq. (4))
$\mathrm{fb}(n)$	決定的フォールバックランキング (eq. (5))
$T_{\max}, C_{\max}$	ステップ・コスト予算
R	ルブリックの木
c_i	葉 i のカテゴリ
w_i	葉 i の重み
judge_i	葉 i の判官
P	論文草稿
$\mathrm{score}(P)$	論文集約スコア (eq. (8))
Refine	改稿演算子 (eq. (9))
ckpt	チェックポイントディレクトリ
$\mathrm{lin}(n)$	n で終端する系譜の連鎖

B　ランタイム LLM プロンプト

本付録は、v0.9.0 時点の `ari-core/ari/prompts/` から逐語複製した、ARI が実行時に使用する LLM プロンプト一覧である。各リストはディスク上のソースとバイト単位で一致している。モデルに渡るバイト列そのものが挙動を決めるため、プロンプトは翻訳せず、本レポートの各言語版で同一の原文をそのまま掲載する。

オーケストレータ

B.1　BFTS 展開

```
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 展開時選択

```
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 選択

```
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 系譜継続判定

```
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 ルートアイデア選択

```
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.
```

エージェントループ

B.6 ReAct エージェントシステムプロンプト

```
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}
```

評価器

B.7 ピアレビュー評価器

```
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 メトリクス抽出

```
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.
```

パイプライン

B.9　キーワードライブラリアン

```
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.
```

ウィザード (Viz)

B.10　ウィザード目標対話

```
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　ウィザード設定生成器

```
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　生成されたサンプル論文

本付録は、section 5.1 のケーススタディ run でシステムが自律的に生成した全 8 ページの論文を、無編集のまま全文収録する。これは成果物としての掲載である——論文が主張するすべての claim は決定論的な claim–evidence ゲートを通過しており、記録された証拠を超える記述には本文中で明示的にヘッジが付されている。ハードウェアは命令セットの水準でのみ記述され、唯一現れる製品名は、引用論文のタイトル内に現れるものであって、run 環境の記述ではない。

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.

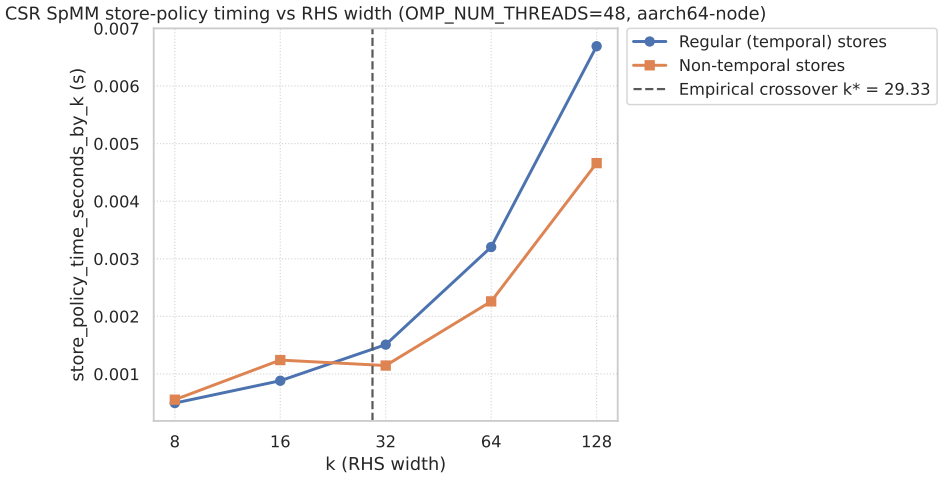

Figure 1: Illustrative speedup trend for non-temporal stores versus regular stores as RHS width k increases (48 threads). This figure is presented as an expected/target behavior motivating the store-policy selector; the specific numerical crossover and speedup values should not be interpreted as artifact-grounded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

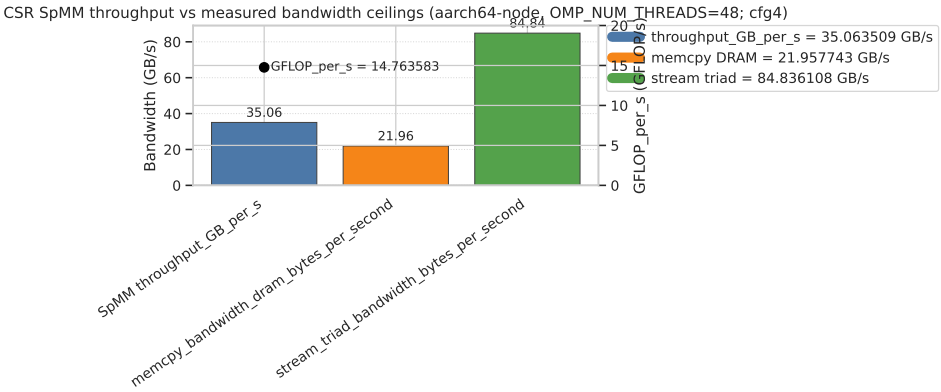

Figure 2: Measured DRAM `memcpy` bandwidth (GB/s) across configurations. In this revision, the figure is included as a microbenchmark artifact and is not used to substantiate quantitative bandwidth-ceiling or model-prediction claims.

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.

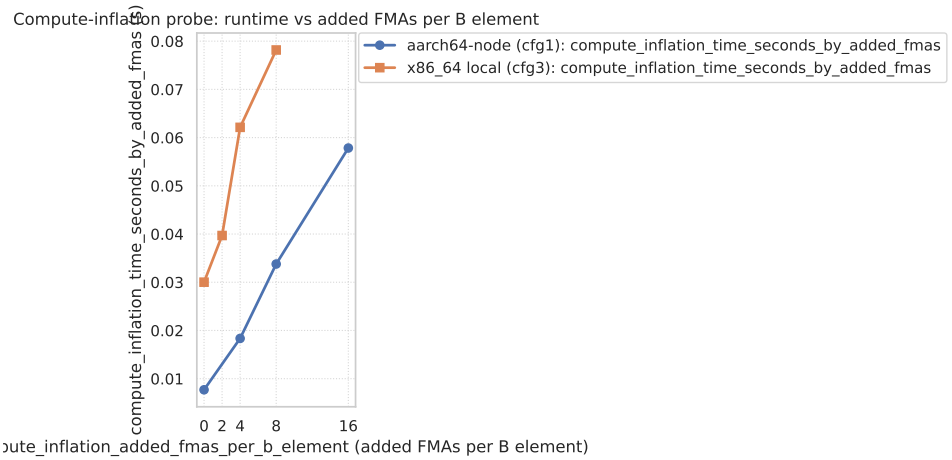

Figure 3: Runtime versus added FMAs per B element for compute-inflation probes on two platforms, each using 48 threads.

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.