ChatGPT

Axplorerと「AIの力をすべての数学者に解き放つ、スパコン不要の無料ツールが登場」記事の深掘り調査レポート

エグゼクティブサマリ

本調査は、MITテクノロジーレビュー[1]日本版の会員限定記事「AIの力をすべての数学者に解き放つ、スパコン不要の無料ツールが登場」（2026年3月31日公開）で提示された主張と、公開コードとしてのAxplorer（GitHub: AxiomMath/axplorer）を一次資料として突き合わせ、技術要素・再現性・導入上の論点を厳密に整理した。[2]

結論として、Axplorer（公開コード）は「離散的最適化（グラフ・点集合など）」を対象に、(1) 例の生成（greedy/局所探索）→(2) 生成例のトークン化→(3) decoder-only Transformer学習→(4) モデルからサンプル生成→(5) CPU並列での復元（detokenize）とスコア計算→(6) 上位例を次世代学習データとして反復、という“PatternBoost系”のループを実装する。[3] その中核は、PyTorchの`scaled_dot_product_attention`を用いた小〜中規模のTransformer実装であり、推論時にはKVキャッシュ（past key/value）を利用し、さらにApple Metal（MPS）環境向けの明示的なメモリ解放（`torch.mps.empty_cache`）がコードに含まれる点が特徴である。[4]

一方、記事の核となる「数万台のマシンで3週間かけた数学問題を、1台のMacが2時間30分で解く」という性能比較は、前提条件（問題サイズN、探索設定、モデル規模、比較対象システム、計測方法）が記事抜粋からは確認できず、厳密なベンチマークとしての再現性は現時点では未検証と判断する（＝“印象的な事例提示”として扱うべき）。[2] ただし、公開実装には「GPU(MPS等)での生成＋CPUプロセス並列でのdetokenize/採点」というパイプライン分離が組み込まれており、同種の探索で壁になりがちな“評価関数のCPUボトルネック”を隠蔽してスループットを上げる設計意図は一次資料から支持される。[5]

学術・実務インパクトとしては、（正式な証明生成ではなく）「反例・最良構成・パターン仮説」の探索を安価に回せる点が、近年の“AI協働による数学研究”潮流（DARPA expMath等）と整合する。[6] ただし適用可能領域は「評価が機械的・高速で、構成をトークン列に落とせる問題」に強く限定され、さらに最終的な正当性保証（証明や形式検証）は別系統のツール（Lean/Isabelle/Rocq等の証明支援系）に委ねる必要がある。[7]

以下の要約表は、ユーザー指定の調査項目（1〜8）を「確認できた事項／未指定」を明示して整理したものである。

調査項目	主要な確認事項（一次資料ベース）	未指定・不足（推測せず明記）
機能・ワークフロー	3つの既定問題（square/isosceles/sphere）＋新問題を“環境（env）”として実装し、反復学習探索を回す。CLIは`python train.py ...`中心。Jupyter notebook（new_envs.ipynb）で拡張手順を案内。	公式GUI（AxplorerとしてのWeb UI）・公式API（REST等）の提供有無は未指定。標準出力物（最良構成の保存形式・場所）の仕様書は未指定。[8]

調査項目	主要な確認事項（一次資料ベース）	未指定・不足（推測せず明記）
技術アーキテクチャ	decoder-only Transformer（PyTorch）＋局所探索（環境依存）を交互に実行。トークン化は座標を単一整数化/座標列/隣接行列など複数。重複排除・正規形化・表現拡張がパラメータ化。GPU生成＋CPU並列採点などの工夫あり。	学習済み巨大モデルの利用・蒸留・量子化の有無は未指定。分散学習（マルチGPU/クラスタ）前提の公式手順は未指定。学習データの外部コーパス利用は未指定（既定は自動生成）。 [9]
性能評価	記事は「数万台×3週間」対「Mac×2時間30分」を主張。公開実装は高スループット化の要素（並列採点、KVキャッシュ、MPSメモリ管理）を含む。	上記比較の測定条件が未指定で、第三者再現のための“ベンチマーク手順・ログ・設定”が不足。公開リポジトリに“公式再現スクリプト／固定seed／評価表”は未指定。 [10]
制約・限界	スコア関数 `calc_score()` の正確性が最重要で、人手監査を強く推奨する手順が明記。探索は「万能薬ではない」旨が記事で強調。	数学的正確性（最終定理の証明）をAxplorer自体が出す、という仕様は未指定。一般的定理証明・自動形式化・証明出力の範囲は未指定。 [11]
学術・実務インパクト	“探索的・実験的”な数学活動を支援する位置づけ。DARPA expMathなどの「数学×AI協働」潮流と整合。	実運用（教育・査読・論文再現性）への制度設計やガイドラインは未指定（各研究コミュニティ側の設計課題）。 [12]
セキュリティ・悪用リスク	ローカル実行中心ならデータ流出面は抑制可能（一般論）。一方で“高速探索”が二面性を持ち得る。	公式の脅威モデル、セキュリティ設計、ログ・通信仕様、ホスト型提供時のSLA/データ取り扱いは未指定。
導入ガイド	micromamba/condaで環境構築し、`train.py` で実行。新問題導入は「実装→ユーザレビュー→スモークテスト」の段階的手順が文書化。	ハード推奨（RAM/GPU/コア数）や費用見積の公式値は未指定（本レポートでは計算法と“例”のみ示す）。 [13]
未解決の疑問点	再現条件、問題ごとの適用限界、保存形式、他ツール連携などが調査優先度高。	—

表の根拠：記事抜粋（会員限定のため可視範囲）および公開コード（README/program.md/model.py/evaluator.py/environment.yml）に基づく。 [14]

調査対象と方法

本レポートの一次資料は、(a) 日本語記事（会員限定。閲覧可能範囲＝冒頭と要点サマリ）と、(b) Axplorer公開リポジトリ（README、依存環境、主要モジュール）である。 [15]　さらにAxplorerが再設計版とされるPatternBoostについては、arXiv論文（アルゴリズム、トークン化、計算例）を参照し、記事主張との整合・相違点を分析した。 [16]

開発元の公式情報としては、アクシオム・マス [17]（表記上はAxiom　Math）の公式サイト上で公開されている「AXLE（Axiom　Lean　Engine）」のドキュメント、ならびに同社サイトのソフトウェア記載を確認した（ただし、Axplorer専用の詳細製品ページは一次資料として十分に確認できなかった）。 [18]

競合・関連領域の一次資料としては、形式検証系（Lean/Isabelle/Rocq）公式ドキュメント、DeepMindの数学/定理証明系成果（FunSearch/AlphaGeometry系）、OpenAIの数学支援事例、さらにDARPA expMathの公式ページを参照した。[19]

重要な制約は、Technology　Review日本版記事が会員限定であり、性能比較の条件・技術詳細の多くが本文側にある可能性が高い点である。このため、本レポートでは「未指定」を明確にした上で、公開コードと論文から再現可能性（再現に必要な情報が揃っているか）を評価した。[2]

Axplorerの設計と運用

Axplorerの機能・ワークフロー

記事の可視範囲では、Axplorerは「数学者向け無料AIツール」であり、従来スーパーコンピュータ上で動作していたPatternBoostを「Mac　Pro上で動く」ように再設計したもの、という位置づけが示されている。[20] また、“LLM（チャットボット）と異なり、新しいパターン発見に特化”という論点が要点として掲げられている。[20]

公開コード（README）から確認できる標準ワークフローは、PatternBoost論文と同型の「局所（local）×大域（global）反復」になっている。具体的には、(1)　greedy生成で初期データを作り、(2)　上位例を学習データとして、(3) decoder-only Transformerを一定ステップ学習し、(4) 温度付きサンプリングで候補を生成し、(5) 生成候補を局所探索で修復・改善してスコアリングし、(6) 上位例を選抜して次エポックへ、(7) 重複が増えたら温度を上げて探索性を確保、という反復である。[21]

入力形式は“自然言語の問題文”ではなく、「環境（env）」として実装された離散最適化問題（例：4-cycle-freeグラフ、isosceles-free点集合、no-5-on-a-sphere点集合）と、そのパラメータ（例：`N`）および実行パラメータ（学習ステップ数、サンプル数、温度など）である。[22]　実行インターフェースは基本的にCLIで、例として`python train.py --env_name square ...`が提示されている。[23]

出力については、少なくとも学習済みモデルとオプティマイザのチェックポイントが`args.dump_path`配下に`model.pt`と`optimizer.pt`として保存され、再開時に読み戻す動作が実装されている。[24] ただし「最良構成（best　construction）をどの形式で保存するか」「研究成果として共有するための標準エクスポート形式」などはREADME上で仕様化されておらず、未指定である（＝ソースコード精読と実運転での確認が必要）。[25]

UI/CLI/API観点では、Axplorer公開コードは「Web UI」や「REST API」を明示していない。一方、同社サイトに公開されている別プロダクトAXLEは、Webインターフェース・Python API・CLI・HTTP APIの提供が明確に記載されている（ただしこれはAXLEの機能であり、AxplorerのUI/APIと混同すべきではない）。[26]

ワークフロー要約表

フェーズ	Axplorerで行うこと	主要パラメータ例（公開資料）	成功/失敗の典型
問題定義	envとして問題を実装（DataPoint/Environment）	`N`, 制約判定, `calc_score()`	`calc_score()`の誤りは全結果を破壊（最重要）
初期データ	greedy生成→上位例選抜	`gensize`, `pop_size`	初期データが弱いと学習が進みにくい

フェーズ	Axplorerで行うこと	主要パラメータ例（公開資料）	成功/失敗の典型
学習	Transformerを数ステップ更新	`max_steps`, `batch_size`, `n_layer`, `n_embd`	過学習/マイナー改善で停滞
生成	温度付きでサンプル生成	`temperature`, `inc_temp`, `top_k`	重複と無効例が増える
修復・評価	detokenize→局所探索→スコア	`num_samples_from_model`, `num_workers`	評価が遅いと探索が詰まる
選抜・反復	上位例＋重複排除→次世代へ	`keep_only_unique`, `make_object_canonical`	多様性不足で局所最適に固定

表の根拠：README、program.md、evaluator実装。 [27]

技術アーキテクチャ

モデル種類と推論（生成）手法

公開実装の中核モデルは、decoder-only Transformerである。自己注意はPyTorchの `scaled_dot_product_attention` を呼び出しており（計算カーネル最適化の恩恵を受け得る形）、ブロックはLayerNorm→自己注意→残差→MLP→残差の標準構成で、GELU活性を用いる。 [4] また、生成時（`generate`）はpast key/value（KVキャッシュ）を保持し、最初のステップだけフル系列を流し、その後は直近トークンのみを入力する“増分デコード”で推論計算量を削減する設計である。 [4]

特筆すべきは、デバイスが `mps`（Apple Metal）である場合に各ステップで `synchronize` と `empty_cache` を呼ぶコードが含まれている点で、Mac上での推論時メモリ圧迫をケアした実装である。 [4] これは記事がMac（Mac Pro）での実行を強調する点とも整合する（ただし記事が示す“2時間30分”条件をこの実装が満たすかは、設定・ハード差が未指定のため別途検証が必要）。 [28]

学習データとデータ生成

Axplorer（公開コード）の学習データは、原則として「環境固有のgreedy構成で生成した有効例を大量に作る→スコア上位 `pop_size` を保持」という自己生成型である。 [29] 例えばREADMEは `gensize=10,000,000` の例を示し、データ生成は高コストになり得ること、生成データを“golden data”としてコピーして再利用することを強く推奨している（生成ジョブがデータを上書きし得る旨も明記）。 [23]

PatternBoost論文においても、局所探索で多数の構成を作り、その“良い構成の集合”からTransformerがパターンを学ぶ、という同型のデータ生成思想が述べられている。 [30]

トークン化・表現学習の要点

公開README上で明示されているトークン化は、(a) 座標を単一整数にエンコード（例：グラフの辺(i,j)を `N*i+j`）する方式、(b) 座標をkトークン列で表す方式、(c) 隣接行列（dense adjacency）として表す方式などである。 [23] さらに、重複排除のための `calc_features` や、正規形化（`make_object_canonical`）、同一対象の複数表現を学習に混ぜる `augment_data_representation` が用意されている。 [31]

PatternBoost論文では、Transformerの計算が系列長の二乗にスケールする点を踏まえ、固定長・可変長のトークン化（BPE等）で系列長を縮める重要性が説明されている。 [30] ただし、Axplorer公開READMEがBPE

の採用を明示しているわけではないため、「AxplorerがBPEをどの程度用い、どの問題で有効か」は未指定である（コードのtokenizer実装確認が追加調査事項）。[32]

並列化・キャッシュ・メモリ戦略

Axplorerの高速化上の中心的工夫は、「GPU（またはMPS）でのサンプル生成」と「CPU側でのdetokenize＋局所探索＋スコア計算」を分離し、CPU側を`ProcessPoolExecutor`（最大`min(20, args.num_workers)`）で並列化しつつ、キューとスレッドで“生成中に評価を進める”パイプライン化を行っている点にある。[33] これは、探索型アルゴリズムで頻出の“評価関数がボトルネック化してGPUが遊ぶ”問題を緩和する設計と解釈できる（推論）。[33]

また、生成側ではKVキャッシュで計算再利用を行い、MPS環境では明示的なキャッシュ解放を行う。[4] これらは“スパコン不要”の主張を支えるための、ワークステーション最適化（特にMac/Apple Silicon含む）として合理的である（推論）。[34]

性能評価・再現性

記事の主張と、一次資料で確認できる範囲

記事の最も強い主張は「数万台のマシンで3週間かけた数学問題を、1台のMacが2時間30分で解く」という時間比較である。[20] しかし、記事抜粋の時点では以下が未指定である：

- どの問題インスタンス（例：`N`の値、制約の定義の細部）を比較しているか（未指定）[2]
- PatternBoost側の「数万台・3週間」が、どの実装・どの探索設定・どの停止条件・どの“成果（最良構成／反例／スコア）”を指すか（未指定）[35]
- Axplorer側の「Mac・2時間30分」が、どのハード仕様（CPU/GPU/RAM）、どの設定（`gensize/pop_size/max_steps/num_samples_from_model`等）で測られたか（未指定）[2]

よって、第三者が同じ条件で“性能差”を検証するための情報は不足しており、現時点では「ベンチマーク結果」というより「象徴的なデモ数値」として慎重に解釈すべきである。[20]

時間比較の棒グラフ（記事提示値）

以下は記事が提示した時間値を、そのまま“時間（hours）”に換算して可視化したものである（3週間＝21日＝504時間、2時間30分＝2.5時間）。**前提条件が未指定であるため、厳密な性能比較ではなく記述の可視化**に留まる。[20]

```
xychart-beta
    title "記事が提示した計算時間の比較（条件は未指定）"
    x-axis ["数万台クラスタ（PatternBoost相当）", "1台のMac（Axplorer）"]
    y-axis "時間（hours）" 0 --> 504
    bar [504, 2.5]
```

（解釈の注意）上記2点だけを単純比で見ると約202倍の時間短縮に見えるが、比較対象の実装・計測・停止条件が不明なため、科学的には“再現可能な主張”としては未確定である。[2]

参考となる別の時間記述（PatternBoost論文）

PatternBoost論文は、（例として）あるサブタスクに関して「単一CPUスレッド500秒、GPU23秒」といった加速例を記述している。[30] これはAxplorer記事の“クラスタ vs Mac”比較とは粒度も対象も異なるが、

「評価（例：多数の行列式計算）をGPU化すると支配的コストを劇的に圧縮できる」こと自体は一次資料で裏付けられる。 [30]

再現性の評価

再現性（第三者が同様の成果に到達できるか）の観点では、Axplorerはコードが公開され、依存環境（environment.yml）や実行例（コマンド）が提示されている点で再現可能性の土台はある。 [36] しかし、(a) 乱数シード管理、(b) 公式の再現実験ログ、(c) “この記事の2時間30分”を再現するための固定設定、(d) ベンチマーク表（複数条件・統計的ばらつき込み）といった科学的再現性を担保する要素は、公開範囲からは確認できず未指定である。 [37]

このため、現段階の実務的な推奨は「Axplorerを“探索支援の研究コード”として扱い、成果を論文やレポートで主張する場合は、設定・seed・ログ・出力物（最良構成）を同梱して自前で再現可能性を作る」ことである（推奨＝本レポートの結論）。 [38]

制約・限界とリスク

制約・限界

Axplorer公開ドキュメントは、新規問題の実装ガイドにおいて `calc_score()` の正確性を「最も重要で、ユーザが監査する関数」と明言し、曖昧点は推測せず確認するよう指示している。 [39] これは、Axplorerが“数学的真偽を証明する装置”ではなく、評価関数を頼りに探索して“良い構成”を見つける装置であることを意味する。 [40]

したがって限界は明確で、(a) 評価が高速で機械的にできる問題ほど向く一方、(b) 評価が遅い／定義が連続的／構造が複雑でトークン化が難しい問題では性能が出ない可能性が高い。これはPatternBoost論文でも、問題によって性能が大きく変動すること、局所探索設計に依存することが述べられている。 [30]

スケーラビリティ面では、Transformer部は系列長に伴う注意計算が支配的になり得るため、トークン化で系列を短くする工夫が本質的になる。 [41] また、サンプル生成数（ `num_samples_from_model` ）や初期生成数（ `gensize` ）を大きくすると、CPU側のdetokenize・採点が計算資源を圧迫する。公開実装はProcessPoolで並列化するが、単一ホストのコア数・メモリ帯域の上限は残るため、問題スケール拡大の限界は“消える”わけではない。 [42]

ライセンスはApache-2.0が明示されており、研究用途の利用障壁は低い。 [23] ただし、生成したデータセット（例：10M例）や成果物の共有ライセンス、第三者データの取り込み方針は未指定であり、共同研究・公開時はチーム内での整理が必要になる。 [23]

セキュリティ・悪用リスクと対策

Axplorer自体は「離散構成探索」を高速化する道具であり、直接的に危険機能を提供するものではない（公開資料から確認できる範囲の事実）。 [37] 一方で一般論として、「探索が安価になる」ことは二面性を持ち得る。たとえば探索対象が暗号・セキュリティ設計に関係する構造（グラフ構成、組合せ設計、SAT/制約充足の強化など）に及ぶ場合、攻撃側にも有利な発見を加速し得る（可能性の指摘＝分析）。 [43]

現実的で低コストの対策は、(a) ローカル実行を基本とし、未公開データを外部送信しない、(b) 新規環境実装での `calc_score()` を複数実装（ナイーブ版と高速版）でクロスチェックする、(c) 生成物の公開前に第三者レビューと再計算で検証する、である（推奨＝本レポートの結論）。 [44]

なお、Axiom側がホスト型サービスとしてAxplorer相当を提供する場合のデータ取り扱い・監査ログ・アクセス制御は、公開資料からは未指定である（AXLEについてはAPI/公開デプロイ運用が文書化されているが、Axplorerに同等の記載は確認できない）。[45]

学術的・実務的インパクト

記事抜粋では、数学は「探索的で実験的」であり、解答発見だけが仕事ではないという問題意識が示されている。[20] Axplorer（PatternBoost系）の強みは、証明や自然言語での解答生成ではなく、「反例候補・最良構成・パターン仮説」を大量に生成し、数学者が次の理論化へ進む材料を得る点にある。[46]

この方向性は、DARPAのexpMathが掲げる「純粋数学の進歩を桁違いに加速するAI共同研究者（AI co-author）」というビジョンとも整合的である。[47] expMath公式説明は、AIが補題分解などを支援し、数学研究のボトルネックを緩和する狙いを明示している。[48]

一方、学術的インパクトを実際に出すには「検証の規範」を崩さない設計が重要になる。ここで有力な補完関係にあるのが形式検証（証明支援）系である。例えばLeanは「最小カーネルが証明項を検査する」ことを信頼性の中心に据える設計が公式マニュアルに記されている。[49] Isabelleも公式サイトで“proof assistant”としての役割を説明し、ドキュメント体系を整備している。[50] またCoq系は2025年にRocqへ改称し、公式にその移行が記録されている。[51]

この文脈でAxiomが別途公開しているAXLEは、Leanコードの検査・証明検証などをWeb/Python/CLI/HTTP APIで提供する、と明確に述べており、探索（Axplorer）→形式検証（Lean＋AXLE）の研究パイプラインを志向している可能性がある（ただし両者の公式統合仕様は未指定）。[52]

競合的枠組みとの比較では、DeepMindのFunSearchは「LLMでコード候補を提案し、評価器が誤りをふるい落としながら反復で改善する」方式として公式に説明され、Nature論文としても公開されている。[43] これはAxplorer/PatternBoost系と同じく「評価器の強さ（機械検証可能性）」を核にする点で近いが、出力が“コード”という形式である点が異なる。[53]

またDeepMindのAlphaGeometry系は、幾何問題を高い水準で解くシステムとして公式発表・Nature論文化されているが、これは“問題の形式表現（formalization）”や推論エンジンとの組合せに強く依存し、Axplorerの「離散構成探索」とは対象が異なる。[54] OpenAI側も数学支援の事例を公開し、研究加速のケーススタディを提示しているが、LLMは誤り（幻覚）を内包し得るため、最終責任は人間にあるという立場を明示している。[55]

総じて、Axplorerが最大の価値を発揮するのは「計算実験で“物体（構成）”を発見し、そこから人間が定式化・証明へ進む」局面であり、教育・査読・検証プロセスでは「出力物（構成）と再現ログ」をセットで提出し、形式検証を組み合わせる運用が望ましい（推奨＝本レポートの結論）。[56]

導入ガイド

研究者が自分の環境で試すための手順

公開READMEが提示する最短導入は以下である。

まず依存環境は`environment.yml`にまとまっており、micromambaで環境を作ってactivateする手順が示されている。[36] 依存にはPyTorch（`torch==2.9.1`）、numba、numpy、scipy、sympy、networkx等が含まれる。[57]

次に、既定環境（例：Turan 4-cycle相当の`square`）を回すコマンド例がREADMEにある。[23] ここでは`N`（問題サイズ）、`encoding_tokens`（表現方式）、`max_len`（系列長上限）、`temperature`/`inc_temp`（探索性）などが主要チューニング点である。[23]

データ生成が高コストな場合は「データ生成だけ」を行い、後で学習に使う方式が推奨されている（`--data_generation_only true`、`gensize`を大きくする）。[23]

中断からの再開は、同一設定＋`exp_id`指定で行い、内部では`dump_path`に保存した`model.pt`/`optimizer.pt`をロードして復元する。[25]

新しい数学問題を載せる手順

新問題は「環境（env）」として実装する。program.mdは段階的に、(1)　問題理解（対象・制約・スコア・トークン化）→(2) 実装（DataPoint/Environment/Tokenizer）→(3) ユーザレビュー（`calc_score`監査を最重視）→(4) スモークテスト→(5) 本番探索、という運用を推奨する。[39]

new_envs.ipynbの冒頭も、既定3問題の説明と、新環境を`src/envs/`に追加しDataPoint/Tokenizer/Environmentを作る手順を示している。[58]

実務上は、**高速化より先に正しさ**を確立する必要がある。`calc_score()`が誤っていると“高スコアに見えるが無効な構成”を大量生成して研究を破壊するためである。[40]

推奨ハードウェア

公式に“推奨スペック表”は提示されていない（未指定）。[37]　ただし公開実装は、(a)　サンプル生成にGPU/MPSを使える設計、(b)　CPUでのProcessPool並列採点を使う設計であるため、経験的には「GPU（またはApple SiliconのMPS）が使えるマシン＋十分なCPUコア数」が効く可能性が高い（推奨＝分析）。[59]

記事はMac　Proでの実行を前面に出しているが、実際にどのMac構成で2時間30分を達成したかは未指定である。[20]

コスト見積もり

ソフトウェア自体はApache-2.0で無償利用できる。[23]　コストは主に計算資源（ローカルなら電力と機会費用、クラウドならGPU/CPU時間）になるが、公式の費用見積は未指定である。[2]

本レポートとしては、再現可能な形での見積方法を提示するに留める（例示値は仮定）：

- ローカル電力コスト（仮定）：`電力(kW) × 実行時間(h) × 電力単価(円/kWh)`
- クラウドコスト（仮定）：`インスタンス単価(円/h) × 実行時間(h)`

記事に出る2.5時間という短時間ケースが再現できるなら、探索1回あたりの直接費は小さくなる可能性があるが、これを一般化する根拠は不足している（未指定）。[20]

トラブルシューティング

公開実装から読み取れる典型的な障害と回避策は次の通りである。

- メモリ不足（特にMPS）：生成ループ内で`torch.mps.empty_cache()`が呼ばれていることから、MPS環境でメモリが逼迫しやすい前提がある。対策としては`batch_size`・`n_embd`・`n_layer`・`max_len`・`gen_batch_size`・`num_samples_from_model`の縮小が筋が通る（推奨＝分析）。 60
- 評価が遅い：evaluatorはProcessPoolで並列化する設計なので、`num_workers`を適切化し、局所探索やスコア計算にnumbaを使うことが推奨されている。 61
- 再現できない（結果がぶれる）：乱択が多い探索であるため、実験ログ、設定、seed（もし導入するなら）を固定し、スモークテストから段階的にスケールする方針がprogram.mdに沿う。 40

未解決の疑問点と追加調査推奨

以下は「未指定だが、科学的検証と実務導入のために重要」な論点を優先度付きで列挙する（推奨＝本レポートの結論）。

優先度	未解決の疑問点	追加調査の具体案
高	“3週間（数万台） vs 2.5時間（Mac）”の比較条件（N・設定・成果定義・停止条件）	公式/チームが再現ログ（コマンド・yaml・seed・ハード構成・成果物）を公開できるか確認。公開できないなら、第三者が公開コードで再現レポを作る。 2
高	出力物（最良構成）の保存形式・所在・再利用手順	実行後ディレクトリ構造とログを特定し、成果物エクスポートを標準化（例：json/csv、証明支援系への変換）。 62
高	tokenization戦略の網羅（BPE等の有無、問題ごとの有効性）	tokenizers実装を精読し、問題×表現×系列長×性能の比較表を作る。 63
中	乱数・再現性（seed管理、統計的評価）	実験プロトコル（複数seed×複数反復）と、スコア分布・到達率を公開する枠組みを作る。 21
中	他ツール連携（例：Lean/AXLE）	発見した構成→命題化→形式証明への“橋渡し”テンプレートを設計（ただし統合仕様は未指定）。 64
中	セキュリティ/ガバナンス（ホスト型提供時）	もしSaaS提供があるなら、データ取り扱い・監査ログ・権限管理を文書化しているか確認。未公開ならローカル運用を前提にする。 65

結論と実務的推奨

Axplorer（公開コード）は、PatternBoost系の「評価可能な構成探索」を、単一マシンで回すための実装上の工夫（GPU/MPS生成＋CPU並列採点、KVキャッシュ、MPSメモリ管理）を備えた“研究用探索基盤”として評価できる。 66　ただし、記事が提示する劇的な時間比較は条件未指定であり、現時点では第三者が同等の主張を科学的に支持する材料が不足している。 2

実務的には、(1) `calc_score()`の厳密監査と単体テスト、(2) 小規模スモークテスト→段階的拡大、(3) 設定・ログ・成果物の保存と共有、(4) 最終成果は証明支援（Lean/Isabelle/Rocq等）で検証、という運用を前提にすると、探索支援ツールとしての費用対効果が最大化しやすい（推奨）。 67

主要ソース一覧

（ユーザー要件に合わせ、URLを明示する。リンク先の内容は本文中で該当箇所を引用・参照した。）

[記事]
https://www.technologyreview.jp/s/379980/this-startup-wants-to-change-how-mathematicians-do-math/

[Axplorer（公開コード）]
https://github.com/AxiomMath/axplorer
https://github.com/AxiomMath/axplorer/raw/refs/heads/main/README.md
https://github.com/AxiomMath/axplorer/blob/main/environment.yml
https://github.com/AxiomMath/axplorer/blob/main/program.md
https://github.com/AxiomMath/axplorer/blob/main/src/models/model.py
https://github.com/AxiomMath/axplorer/blob/main/src/evaluator.py
https://github.com/AxiomMath/axplorer/blob/main/src/trainer.py

[Axiom Math（開発元）公式]
https://axiommath.ai/
https://axiommath.ai/software
https://axle.axiommath.ai/v1/docs/

[PatternBoost（関連論文）]
https://arxiv.org/html/2411.00566v1

[DARPA expMath（背景）]
https://www.darpa.mil/research/programs/expmath-exponential-mathematics
https://www.darpa.mil/news/2025/math-ai-tomorrows-breakthroughs

[形式検証・定理証明（競合/補完ツールの公式資料）]
https://lean-lang.org/doc/reference/latest/
https://isabelle.in.tum.de/
https://isabelle.in.tum.de/documentation.html
https://rocq-prover.org/releases/9.0.0

[DeepMind（数学AIの代表例）]
https://deepmind.google/blog/funsearch-making-new-discoveries-in-mathematical-sciences-using-large-language-models/
https://www.nature.com/articles/s41586-023-06924-6
https://deepmind.google/blog/alphageometry-an-olympiad-level-ai-system-for-geometry/
https://www.nature.com/articles/s41586-023-06747-5
https://deepmind.google/blog/advanced-version-of-gemini-with-deep-think-officially-achieves-gold-medal-standard-at-the-international-mathematical-olympiad/

[OpenAI（数学支援の公式事例）]
https://openai.com/index/gpt-5-mathematical-discovery/
https://openai.com/index/gpt-5-2-for-science-and-math/

1 3 8 9 13 21 22 23 25 27 29 31 32 36 37 63 66 raw.githubusercontent.com
https://github.com/AxiomMath/axplorer/raw/refs/heads/main/README.md

2 10 12 14 15 20 28 35 46 MIT Tech Review: AIの力をすべての数学者に解き放つ、スパコン不要の無料ツールが登場
https://www.technologyreview.jp/s/379980/this-startup-wants-to-change-how-mathematicians-do-math/

4 34 59 60 axplorer/src/models/model.py at main · AxiomMath/axplorer · GitHub
https://github.com/AxiomMath/axplorer/blob/main/src/models/model.py

5 33 42 61 axplorer/src/evaluator.py at main · AxiomMath/axplorer · GitHub
https://github.com/AxiomMath/axplorer/blob/main/src/evaluator.py

6 47 expMath: Exponentiating Mathematics
https://www.darpa.mil/research/programs/expmath-exponential-mathematics?utm_source=chatgpt.com

7 19 49 The Lean Language Reference
https://lean-lang.org/doc/reference/latest/?utm_source=chatgpt.com

11 17 38 39 40 44 56 67 axplorer/program.md at main · AxiomMath/axplorer · GitHub
https://github.com/AxiomMath/axplorer/blob/main/program.md

16 30 41 PatternBoost: Constructions in Mathematics with a Little Help from AI
https://arxiv.org/html/2411.00566v1

18 Axiom
https://axiommath.ai/

24 62 axplorer/src/trainer.py at main · AxiomMath/axplorer · GitHub
https://github.com/AxiomMath/axplorer/blob/main/src/trainer.py

26 45 52 64 AXLE Documentation
https://axle.axiommath.ai/v1/docs/

43 53 FunSearch: Making new discoveries in mathematical ...
https://deepmind.google/blog/funsearch-making-new-discoveries-in-mathematical-sciences-using-large-language-models/?utm_source=chatgpt.com

48 Math + AI = Tomorrow's breakthroughs
https://www.darpa.mil/news/2025/math-ai-tomorrows-breakthroughs?utm_source=chatgpt.com

50 Isabelle
https://isabelle.in.tum.de/?utm_source=chatgpt.com

51 Rocq Prover 9.0.0 Release Notes
https://rocq-prover.org/releases/9.0.0?utm_source=chatgpt.com

54 AlphaGeometry: An Olympiad-level AI system for geometry
https://deepmind.google/blog/alphageometry-an-olympiad-level-ai-system-for-geometry/?utm_source=chatgpt.com

55 Advancing science and math with GPT-5.2
https://openai.com/index/gpt-5-2-for-science-and-math/?utm_source=chatgpt.com

57 axplorer/environment.yml at main · AxiomMath/axplorer · GitHub
https://github.com/AxiomMath/axplorer/blob/main/environment.yml

58 raw.githubusercontent.com
https://github.com/AxiomMath/axplorer/raw/refs/heads/main/new_envs.ipynb

65 Axiom AI

https://axiommath.ai/territory/axplorer?utm_source=chatgpt.com