LLM支援によるプラズマシミュレーションコード自動化：ドラフト2

関太郎（seki.kentaro.66s@st.kyoto-u.ac.jp） 京都大学 *日付：本日*

要旨

目次

1. はじめに

1.1 動機

現代の核融合研究は、**ジャイロ運動論（gyrokinetic）シミュレーション**に大きく依存している。これは、本来6次元で表される運動論的プラズマ問題を、重要な微視的乱流物理を保ちつつ扱いやすい形へと縮約する手法である。この転換により、実験的には得難い「第一原理」乱流研究や輸送予測が可能になった。現在では、トカマク／ステラレータ乱流や検証研究において、大規模なジャイロ運動論コード（例：GENE、GS2、GTC、GKVおよび関連する検証研究）を用いることは日常的である。(MPG.PuRe)

しかし、**ジャイロ運動論シミュレーションソフトウェアは拡張が難しい。**

- **大規模かつ相互依存の強いコードベース**：衝突、電磁項、幾何、診断などの物理モジュールが、多数のファイルと抽象化層に分散している。
- **新機能追加あたりのコストが高い**：一見小さな物理変更（例：新しい衝突演算子の項、平衡インタフェースの追加、新しい診断量の追加）でも、データ構造、ソルバ、I/O、テスト、ドキュメントなど複数箇所の修正が必要になりがちである。
- **オンボーディングの負担が大きい**：新規貢献者は、物理とコードアーキテクチャの双方を学んだうえで、安全な変更ができるようになる必要がある。

近年、大規模言語モデル（LLM）は、**外部知識・ツール・構造化されたエージェント型パイプライン**と組み合わせることで、ソフトウェア開発や科学ワークフローにおいて実用的な道具になりつつある。これを踏まえ、本研究では核融合研究者が（1）未経験のコード領域をより速く理解し、（2）変更をより確実に実装できるよう支援するLLMシステムの構築を目指す。最終的には、シミュレーション研究における反復サイクルの加速が期待される。

1.2 大規模言語モデルの近年の発展

ここ数年の進展により、現代のLLMシステムが、単純でない科学計算コードにも有効に働き得る理由が説明できる「技術スタック」が形成された。

(1) Transformer

現代のLLMの多くは、再帰を**自己注意（self-attention）で置き換えたTransformer**に基づく。直感的には次の通りである。

- RNNでは、情報は時間方向に逐次的に流れる必要がある。
- Transformerでは、各トークンが注意機構によって他のトークンを直接参照でき、**並列学習**が可能になり、長距離依存の扱いも改善される。

このアーキテクチャにより、最適化の安定性を保ちつつ、非常に大規模なデータセットとモデルへ学習をスケールさせることが可能になった。(arXiv)

(2) スケーリング則（および計算最適学習）

スケーリング則の研究は、モデルサイズ・データ量・計算量を増やすと、損失がしばしば**滑らかなべき乗則**に従うことを示し、進歩が予測・設計可能であることを明らかにした。(arXiv) さらに後続研究では、固定計算予算の下で、多くのLLMがトークン数が少なすぎるために「学習不足」であることが示され、**計算最適（compute-optimal）**な学習戦略（いわゆる「Chinchilla」の視点）が提案された。(arXiv)

科学計算コードにとって重要なのは、「大きいほど良い」だけでなく、計算資源を効率よく配分する方法論が得られる点である。これは後に述べる、RL系の後学習に伴う計算コストの議論にも直結する。

(3) InstructGPT / RLHF（PPO系が多い）

ベースLLMは次トークン予測を目的に学習されるため、必ずしも指示に従うようには設計されていない。
InstructGPT系のパイプラインは、これを実用化するレシピを与えた。

1. 人手による**デモ（教師ありデータ）**を収集し、教師あり微調整する。
2. モデル出力に対する**人手の好み順位**を収集する。

3. それを予測する**報酬モデル**を学習する。
4. 報酬を改善しつつベースモデルからの乖離を抑えるよう、RL（多くは**PPO**）で方策を最適化する。

この流れは、指示追従性とユーザ嗜好との整合を改善し、場合によっては「はるかに大きいベースモデル」より「小さい整合済みモデル」の方が好まれることさえ示した。(arXiv) また、この「好みから学ぶ」枠組み自体は、RL分野でも確立されたテーマである。(arXiv)

(4) DPO（Direct Preference Optimization）

DPOは、好み最適化を**分類に近い単純な目的関数**として捉え直し、RLHFに近い目的（KL制約下での報酬最大化）を、完全なRLロールアウトなしで達成できることを示す。実務上は次の利点がある。

- 学習時に明示的な報酬モデルを必要としない（「暗黙的」になる）。
- 学習が教師あり微調整に近くなり、安定性や実装負担が軽くなることが多い。

本研究では、学術的GPU予算でも現実的な学習ループを目指すため、この点が重要である。(arXiv)

(5) DeepSeek-R1系の推論強化後学習

近年の推論重視モデルの公開や報告（例：DeepSeek-R1）では、より強い推論挙動を得る手段として、**強化学習／好み最適化**への再注目が示されている。ここでは、データ設計と評価設計を丁寧に行うことと組み合わされることが多い。(arXiv)

本稿で重要なのは「どのモデルが最良か」ではなく、分野全体の傾向として、**後学習（好み、RL、あるいはDPOのようなRL不要の変種）**が推論力と信頼性の主要なレバーとみなされつつある、という点である。

(6) テスト時計算（推論時探索）

重みを更新しなくても、推論時に**より多くの計算**を使うことで推論性能を改善できる場合がある。代表的なパターンは次の通り。

- **Chain-of-thoughtプロンプト**：中間ステップを書かせる。(OpenReview)
- **Self-consistency**：複数の推論トレースをサンプルし、多数派の答えを採用する。(Astrophysics Data System)
- **Tree of Thoughts（ToT）**：部分解の木を探索し、分岐評価とバックトラックを行う。(OpenReview)

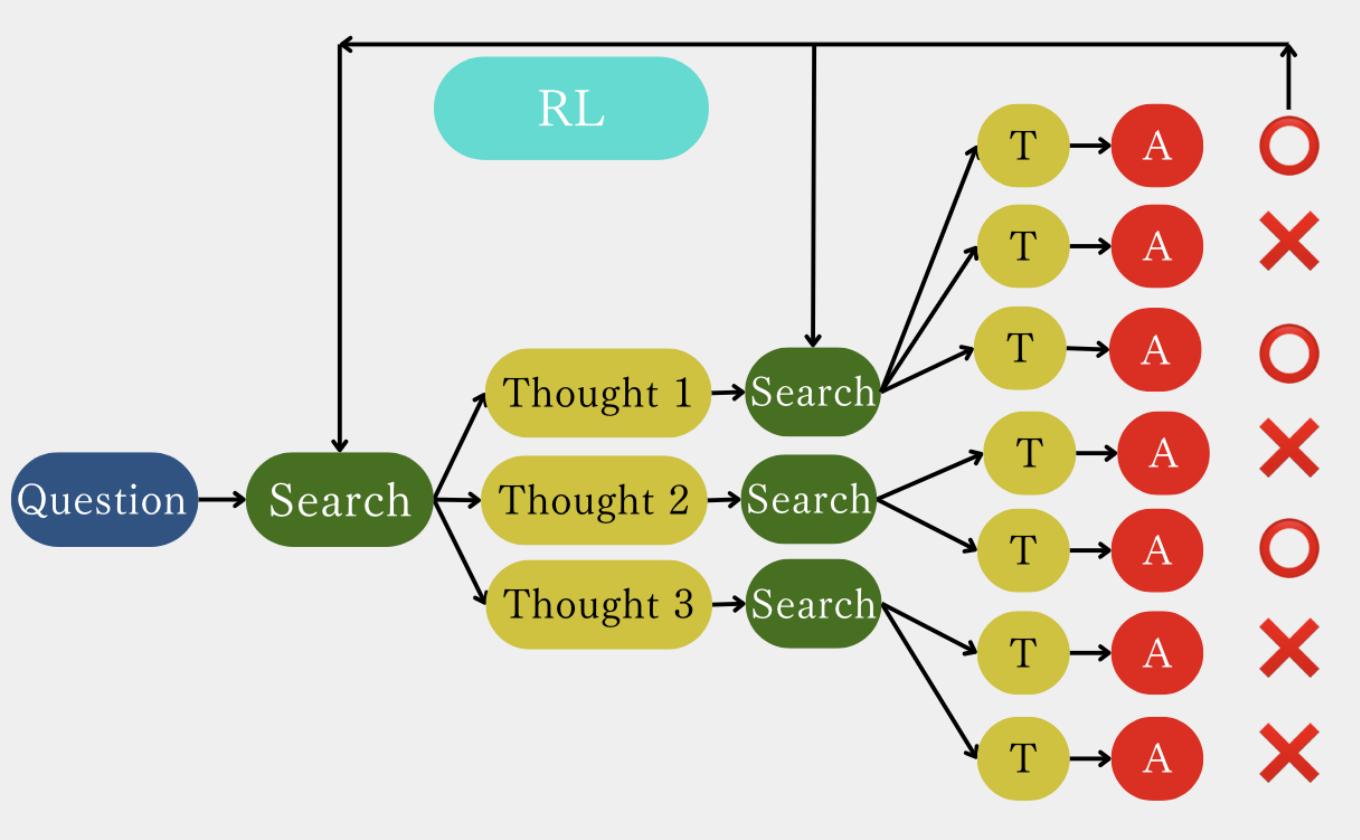
直感的には、最初の下書きをそのまま信じず、複数の候補（「証明の試み」）を生成して、最も良いものを選ぶ／洗練する、というアナロジーである。この「テスト時探索」は、複数のパッチ候補を生成して採点するエージェント型コーディングパイプラインと直接結びつく。

RLが有効な理由と、その高コスト性

経験的に、好み学習／RL型の後学習は、**指示追従性**を改善し、ときに**推論ベンチマーク性能**も向上させる。しかし、（PPOなどで）多量のサンプル生成（ロールアウト）と複数回の最適化ステップが必要になり、資源コストが高い。(arXiv) このため、適切な場合には、DPOのような**RL不要**（または教師ありに近い）代替を検討する動機がある。(arXiv)

1.3 SEIMEI

本研究では、LLMベース推論を**検索統合型のエージェント・パイプライン**としてオーケストレーションするオープンソースライブラリ**SEIMEI**を用いる。概念的にSEIMEIは、ドメイン固有知識と推論を要するタスクを想定して設計されている。SEIMEIは、エージェントと知識を統合する検索モデルを強化学習することで、これを実現する。(GitHub)



図｜SEIMEIの推論・学習パイプライン。検索モデルに対する強化学習により、思考を誘導することで推論経路を改善する。

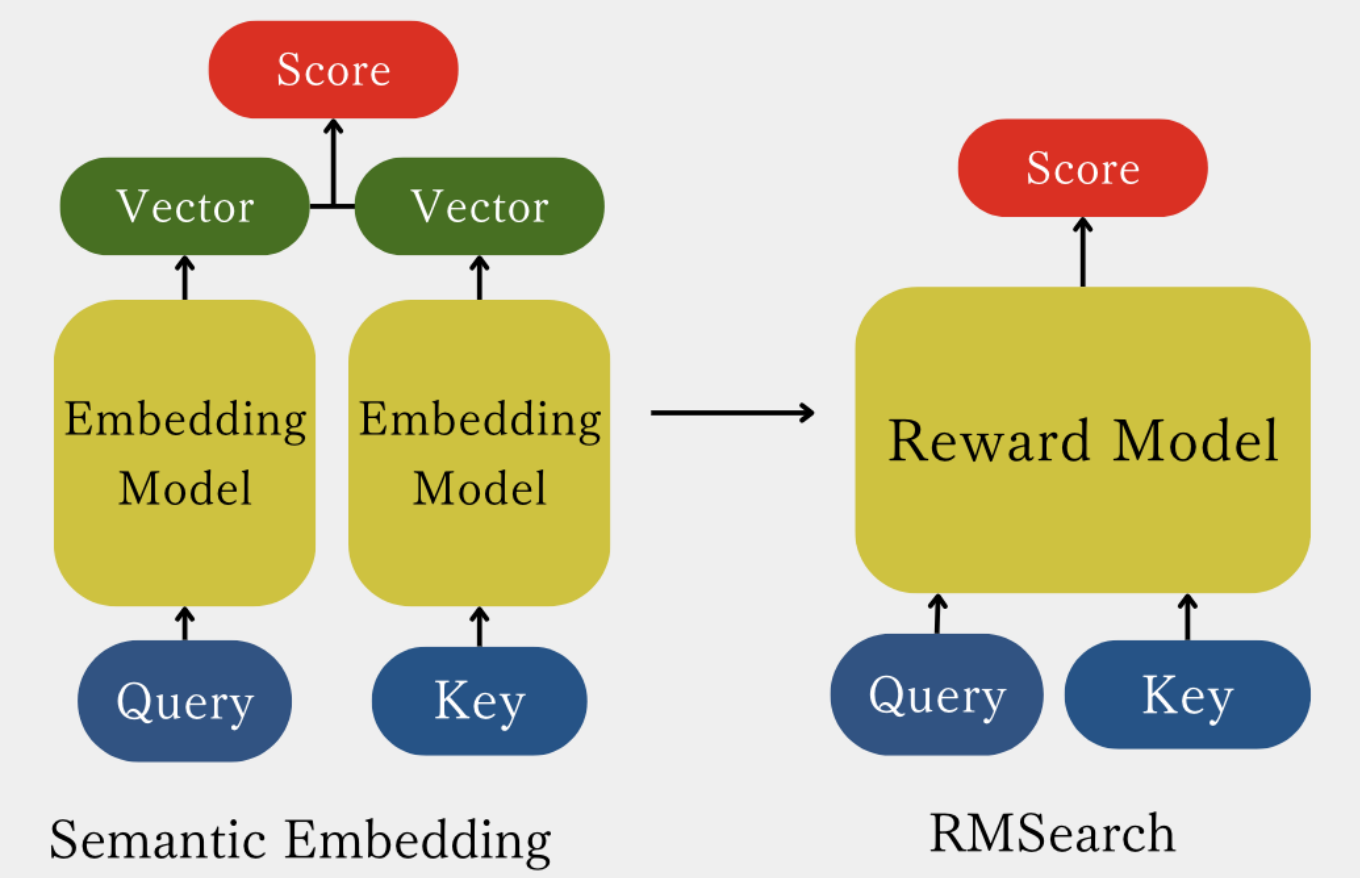
要点（平易に言うと）：SEIMEIは、LLMを次トークン予測タスクで追加学習するのではなく、**推論を誘導する検索モデルを学習する**。この特性は、従来手法に比べ次の利点を持つ。

1. 検索モデルの学習は推論用LLM本体を壊さないため、中核推論の崩壊を防げる（！継続学習の引用を追加する必要あり）。
2. 特定ドメインへの適応は、次トークン生成LLMを学習するよりも、検索モデルの適応の方が計算コストが小さい。

検索は知識拡張の中核技術である。検索エンジンは、市民が作成した膨大な文書を結び付け、人間が自分の知識を追加することで検索システム全体を強化できるようにしてきた。この「知識追加の柔軟性」は、AIシステムにとっても知識拡張の鍵となる。SEIMEIは、単なる知識だけでなく「考え方」まで統合する検索モデルを持ちうる点で、現在のワークフロー型エージェント・パラダイムを超える新しいAIシステムになり得る。

1.4 RMSearch

SEIMEIのルーティングおよび知識選択は、報酬モデル風の検索／再ランキング要素である**RMSearch**によって行われる。RMSearchは、LLMの次のステップを補強するために有用な断片（知識、過去の解法、文書、ヒューリスティクス）を取得し、優先順位付けする。(GitHub)



図｜従来のベクトル検索モデルとRMSearchのモデル構造の違い。

構造的にRMSearchは**再ランキング（reranker）**に最も近い。すなわち、クエリと候補テキストが与えられたとき、クエリー候補の関連性をスコアリングする（しばしばクロスエンコーダ型の構成）。この見方は、BERT型再ランキングや効率的な相互作用モデルなど、長い再ランキング研究の流れと整合する。(arXiv)

実務上の違いは、取得パイプラインにある。多くのシステムでは、まず密な検索器がトップk（例：100）を返し、その短いリストを再ランキングして最終的な少数を選ぶ。RMSearchは、報酬モデルで候補を直接スコアリングするため、選択と同じ信号で関連性を計算でき、単なる後処理として分離されない。

RMSearch（再ランキング）が特に重要なのは、ドメイン固有検索では一般的な埋め込み類似度がタスク関連信号を見落とすことがあるためである。密な検索器はテキストをベクトルに圧縮する一方、報酬モデルはクエリー候補のテキスト対から直接関連性スコアを学習できる。このため、RMSearchは好みデータを用いて特定ドメインに適応させやすい。(GitHub)

1.5 本研究の目的

本研究では、SEIMEI + RMSearchを用いて、LLM支援による**プラズマシミュレーションコードの記述／編集**を改善することを目指す。対象はジャイロ運動論コード（GKV）である。計画は次の通り。

1. SEIMEIを用いて推論システムを構築する。
2. 実際のジャイロ運動論コードベースからデータセットを生成する。
3. タスクを解く／修復する過程から、再利用可能な教訓を抽出し、**知識プール**を構築する。
4. 好みデータ（DPO）を用いて、その知識プール上でRMSearch（再ランキング）を学習する。
5. 知識プールと学習済みRMSearchによって、同一ドメインにおけるコード修復精度が改善するかを測定する。

（あなたの要望に従い、このベースラインドラフトには実験結果を含めない。）

1.6 関連研究

以下に本研究が依拠する主要な研究潮流を示す。本節は意図的に平易かつ学際的に記述する。

Retrieval-Augmented Generation（RAG）

RAGは、LLMに外部メモリ（文書、断片、コード、論文など）を接続し、パラメータだけに頼らず関連コンテキストを**検索して取り込む**ことで生成を支援する方法である。中核の動機はモジュール性にある。

- コーパス／インデックスを更新すれば知識を更新でき、モデル全体を再学習しなくてよい。
- 「この回答が参照したソース」を付与できる（出典性）。

古典的なRAG研究は、知識集約タスクに対してニューラル検索器と生成器を組み合わせる。(arXiv) 関連として、事前学習中の検索（REALM）(arXiv)、超大規模検索による生成強化（RETRO）(arXiv) がある。 また、密検索はオープンドメインQAや検索パイプラインの標準的ベースラインとなった。(arXiv)

本研究で重要な理由：コードベースは「文書」である。核融合コードのリポジトリには、関数契約、データ配置、物理仮定といった地の文が含まれる。RAG型のグラウンディングは、幻覚的編集を減らす。

AIエージェント（ツール利用、ブラウジング、反復編集）

現代の「LLMエージェント」は、言語モデルにツールと反復的な制御を組み合わせる。

- 学習またはプロンプトによるツール利用（例：Toolformer）。(arXiv)
- LMと特化コンポーネントを混ぜるモジュール型アーキテクチャ（MRKL）。(arXiv)
- 多段タスクに有用なreason+act型プロンプト（ReAct）。(GitHub)
- ブラウザ支援による参照収集と根拠付き推論（WebGPT）。(arXiv)
- リポジトリ探索と編集インタフェースを備えたソフトウェア工学エージェント（SWE-agent）。(arXiv)

実務的なエージェント枠組みは、プロンプトだけでなく、ファイル編集、チェック実行、ブラウジングなどの*インタフェース*を重視する傾向がある。これは、構造化されたコード修復を目標とするSEIMEIの設計思想とも整合する。(arXiv)

DSPy（評価駆動の自動パイプライン改善）

DSPyは、LLMパイプラインを宣言的に記述し、評価指標を定義し、コンパイラのような最適化器がデータを使ってプロンプトやモジュールを改善する枠組みを提案した。(arXiv)

本研究との関係：我々も「評価に基づく改善ループ」を狙うが、焦点は（a）コード修復タスク、（b）検索／再ランキング（RMSearch）と知識プールの改善にある。

再ランキングモデル（関連性スコアリング）

再ランキングは情報検索の標準技術である。高速検索器で候補を集め、より強力なモデルで順位付けし直す。BERT再ランキング（monoBERT）は、パッセージランキングで大きな改善を示した。(arXiv) ColBERTは遅延相互作用により、効率と品質のトレードオフを実現する。(arXiv)

本研究との関係：RMSearchは「次にどの知識／コンテキストを使うべきか」を再ランキングする役割を担う。ジャイロ運動論のような専門領域では、これが特に重要になる。

---

2. 手法

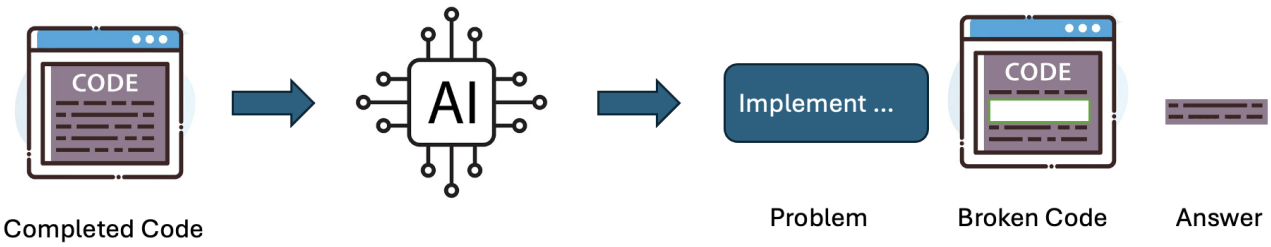
2.1 概要

本研究の手法は、実際の核融合コードベースから（1）コード修復タスク、（2）エージェント型推論パイプライン、（3）ドメイン適応された再ランキング器（RMSearch）の学習データを生成するエンドツーエンドのループである。

1. **データセット作成**：意図的にコードを壊し、その修復を求める質問を生成する。
2. **推論パイプライン**：（質問、破損コード）から修復パッチ候補を生成する。
3. **スコアリング**：候補修復を評価する（静的チェック、diff品質、可能ならテスト／コンパイル）。
4. **知識生成**：成功（および失敗）から再利用可能な「教訓」を収集する。
5. **RMSearch用データセット構築**：知識／コンテキスト候補、あるいは修復候補に対する好みペアを作る。
6. **RMSearch学習**（DPO型の好み最適化）。
7. **評価**：学習済みRMSearchでルーティング／増強した推論パイプラインを再実行する。

本設計は、完全なRLHFインフラなしでも実現可能でありつつ、好み学習の利点を取り込むことを意図している。

2.2 データセット作成



図｜データセット生成パイプライン。LLMに完全なコードを意図的に破壊させ、修復に関する問題を生成する。

対象リポジトリ：データセット生成元として、オープンなジャイロ運動論Vlasovシミュレーションコード **GKV（gkvp）**を用いる。(GitHub)

タスクタイプ（平易な説明）：「コードが本来意図した物理に即した挙動へ戻るように修復する」。

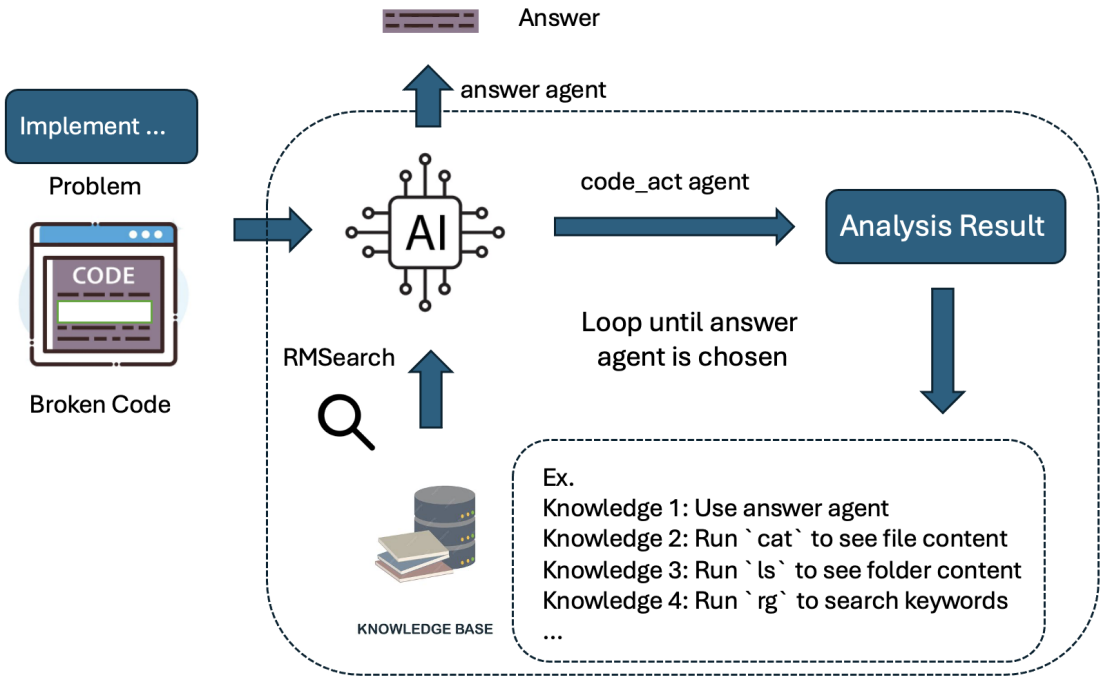
具体的には、あるファイルに対して次を行う。

1. LLMが**壊す行**を選ぶ。制約として、削除はコメントや些末な整形ではなく、**中核的な物理ロジック**に影響する必要がある。
2. パイプラインは次を生成する。
    - 重要な行を削除する**パッチ**
    - 観測される失敗や欠落挙動を記述し（「欠落した計算／境界条件／項を復元せよ」など）、LLMに修復を依頼する**問題文**
3. LLMがパッチのデバッグを行う。パッチは適用時に失敗することが多いため、エラーメッセージに基づいてパッチを数回修正する。それでも失敗するパッチは除外する。

「壊して直す」方式の利点：

- **正解性**：正しい修復は元コードにあるため、正解パッチが明確である。LLM出力が正しいかは元コードとの比較で容易に検証できる。
- **スケーラビリティ**：必要なのは完全なコードとLLMへの依頼だけであり、プラズマ物理の他コード、さらには他分野にも容易に拡張できる。
- **現実性**：研究者が遭遇しがちな、プラズマ方程式に関わる小規模追加・修正を中心とした編集問題を生成できる。

2.3 推論パイプライン



推論ループはSEIMEIで実装する。(GitHub)

GKVコード解析では基本的に2種類のエージェントを使う。

- **code_actエージェント**：ローカルのフォルダ／ファイルを解析するために、Unixコマンド（例：`cat`、`ls`、`rg`）やPythonコードを実行する。
- **answerエージェント**：code_actの解析結果を要約し、最終回答を作る。

これは、ツール利用やリポジトリ規模編集を扱う先行研究で示されたエージェントパターンと整合する。

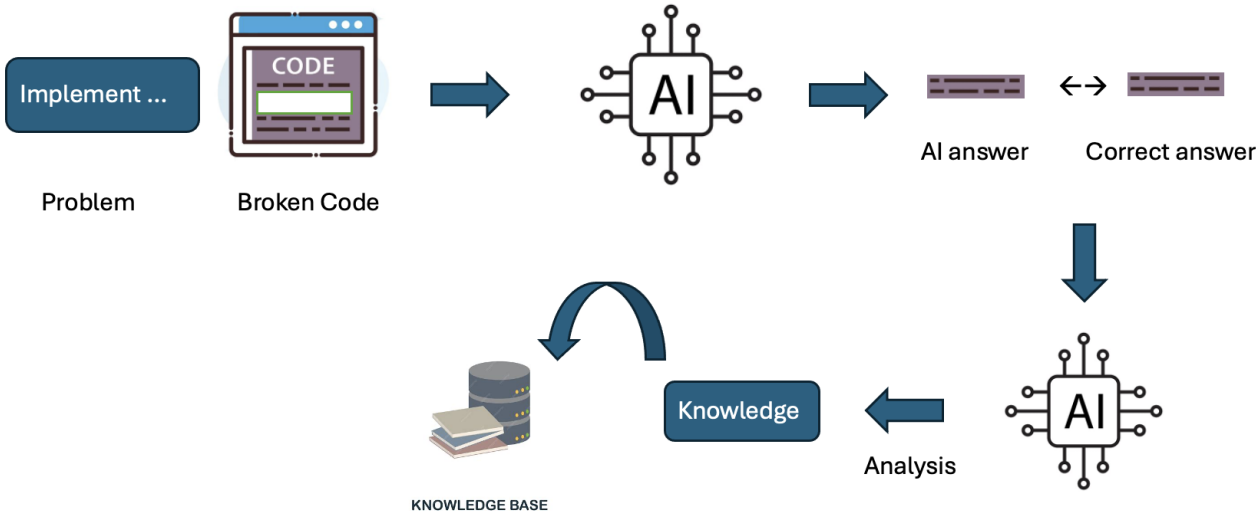
- ReAct型の推論＋行動ループ（段階的に進める）。(GitHub)
- ToolformerやMRKLに代表されるツール拡張・モジュール型。(arXiv)
- SWE-agentは、慎重に設計された「エージェントーコンピュータI/F」が、ベンチマーク上のコード編集成功率を高めることを示した。(arXiv)
- OpenHandsはコーディングエージェントのためのオープンプラットフォーム例である。(GitHub)

これらに対してSEIMEIは、さらに**ルーティングと増強**という特徴を持つ。各ステップでRMSearchを呼び出し、ドメイン関連の知識断片を取得し、それを次のモデル呼び出しに組み込める。(GitHub)

本研究では、従来の意味埋め込み（semantic embedding）モデルではなく、検索モデルとしてRMSearchを用いる。RMSearchは（しばしば再ランキングモデルと呼ばれるように）関連性スコアをより直接に計算でき、意味埋め込みモデルよりも特定ドメインへ適応しやすいからである。(GitHub)

2.4 知識生成

知識は相補的な2つの方法で生成する。



図｜正解からの知識生成パイプライン。

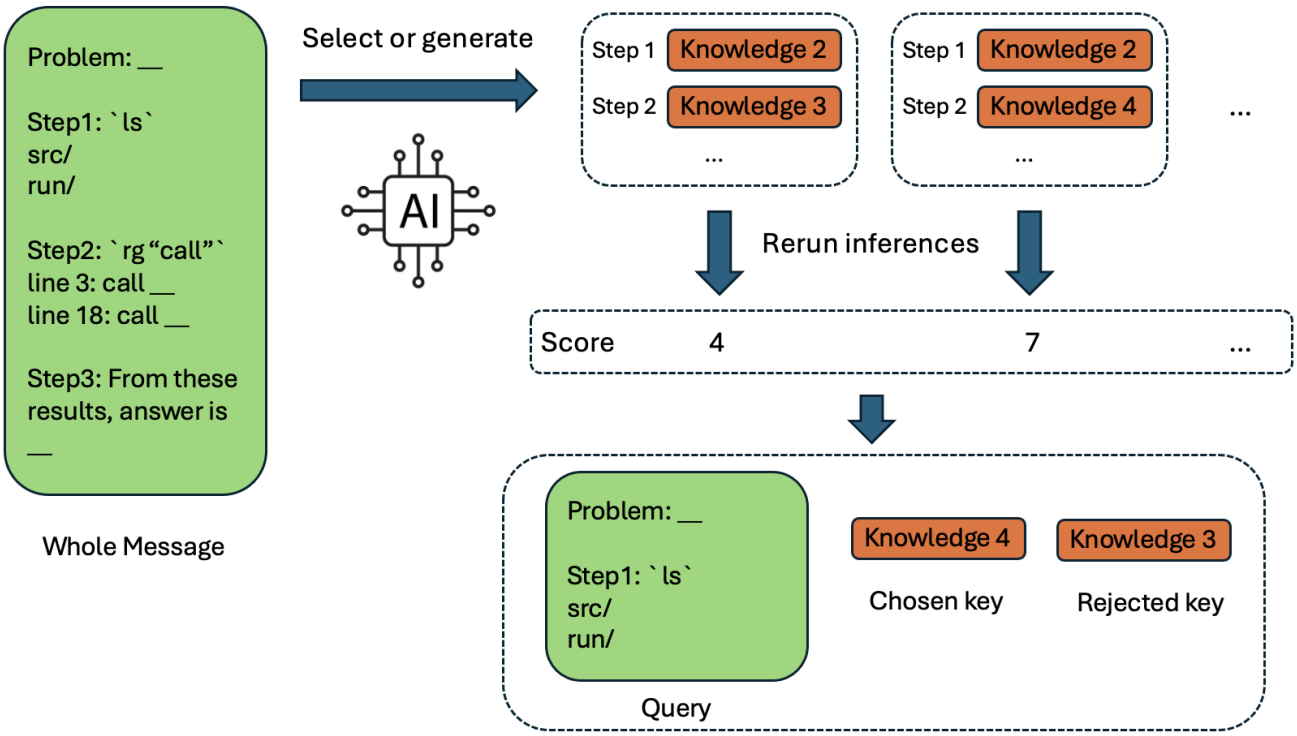
1. **自動知識生成（修復結果から）：** エージェントの出力パッチと期待される正解修復を比較し、再利用可能な文として抽出する。例：
    - 「ジャイロ平均化ポテンシャル項を変更する場合は、___の正規化も更新する。」
    - 「このファイルはフラックスチューブ幾何を仮定しており、境界条件は___で実装されている。」

    目的は、「1つの解けた事例」を将来事例に効くヒントへ変換することである。

2. **手動知識：** ドメイン専門家（または注意深い読者）が、タスクに関する簡潔な指示を書く。例：
    - 「十分なエージェント出力が得られたらanswerエージェントを実行する。」
    - 「現在フォルダの中身を見るために`ls`を実行する。」
    - 「ファイルの中身を見るために`cat`を実行する。」
    - 「フォルダ横断でキーワード検索するために`rg`を実行する。」
    - 「問題を解く戦略を、ファイル比較中心の分析へ変更する。」

自動知識はスケールし、手動知識は高精度になり得るため、両方を保持する。

2.5 RMSearch用データセット



図｜DPO学習のための好み型データセット作成（概念図）。

RMSearchの学習には、好み型データセットが必要である。本研究ではパイプライン実行ログから、次のような比較を作る。

- **知識の好み**：クエリ（例：「欠落した物理項を修復せよ」）と、それまでのエージェント出力が与えられたとき、知識断片AとBを比較し、それを使った場合により高スコアな解が得られる方を選好とする。

これはRLHFで用いられる「好みから学ぶ」一般枠組みに一致するが、本研究では生成モデル自体ではなく、**検索／再ランキングとエージェントのルーティング**に適用する。(arXiv)

この方法では、知識サンプリングと選好付与が鍵になる。本実験で用いるサンプリング手順は以下である。

- **知識生成または選択**：ベース推論試行の全メッセージ履歴から、指定ステップ向けの知識をLLMが生成または選択する。複数ステップにまたがる知識テキスト集合を**knowledge chunk**と呼ぶ。各問題について複数チャンクを作る。
- **knowledge chunkごとに推論を再実行**：各チャンクを用いて推論を再実行し、チャンク当たり複数回の実行を行う。
- **knowledge chunkのスコアリング**：各推論実行を採点し、チャンク単位で平均してスコアを得る。
- **好みデータセットへの変換**：チャンクスコアから（query, chosen knowledge, rejected knowledge）ペアを構築する。ここでqueryは、chosen/rejected知識で増強されるステップまでのメッセージ履歴である。些末な差は閾値で除外する。

2.6 RMSearchの学習

RMSearchは**DPO型の好み最適化**により学習し、再ランキング器が望ましい項目（知識断片やパッチ）に高いスコアを付けるようにする。(arXiv)

具体的にRMSearchは、**query + key**をスカラー報酬／スコアへ写像する関数を実装する：すなわち $s_{\theta}(q, k) \rightarrow r$。好み三つ組 (q, k^+, k^-) に対し、DPO損失を次のように定義する。

$$ \mathcal{L}_{\mathrm{DPO}} = -\log \sigma\left(\beta\left[s_{\theta}(q,k^+) - s_{\theta}(q,k^-)\right]\right) $$

ここで、qはクエリ、k^+は好ましいkey（知識または候補パッチ）、k^-はそれより劣るkey、s_{θ}はRMSearchのスコア（報酬）関数、s_{ref}は更新を保守的に保つための固定参照スコアラ、βは選好マージンの鋭さを制御し、σはロジスティック関数である。これにより、RMSearchとの対応が明確になる。すなわち学習は、**query + key -> reward**のスコアを調整し、望ましい知識／パッチにより高いスコアを割り当てる。これは推論時の再ランキング挙動を直接改善する。

バッチ構成（InstructGPTの報酬モデル学習手順に準拠）：好み学習における実務上の問題として、同一クエリから派生した多数のペアワイズ比較は強く相関しやすい。InstructGPTの報酬モデル学習では、ラベラーが各プロンプトにつき ${K}$ 個（${K}$は4〜9）の候補完了を順位付けし、最大で $\binom{K}{2}$ 個のペアワイズ比較が得られるが、これらを独立データとして単純にシャッフルして学習すると急速な過学習が生じることが報告された。そこで、著者らは**単一プロンプト由来の $\binom{K}{2}$ 比較を、1つのバッチ要素として扱う**。これは（i）候補あたり1回の順伝播で済むため $\binom{K}{2}$ 回より**計算効率が高く**、（ii）相関を保ったまま扱うことで**安定性が高く**（過学習が減り、検証損失／精度が改善する）という利点がある。RMSearchでも同様のバッチ原理を採用する。すなわち、各ミニバッチは(B)個の異なるクエリからなり、各クエリについて ${K}$ 個の候補keyをスコアリングし、クエリ内の比較集合に対してペアワイズ好み損失を適用する。クエリ間で比較を無差別に混ぜないことで、RMSearchが推論時に解くべき「クエリ内再ランキング」構造と整合した更新になる。(arXiv)

2.7 RMSearchと推論の評価

---

3. 実験

3.1 「rmsearch-plasma-1.0」の評価

3.2 「seimei-plasma-1.0」の評価

3.3 人手評価

---

4. 議論

4.1 ドメイン特化LLMの可能性

4.2 うまくいかなかった試み

- サンプリング手法
- Codexでデータセット作成

---

5. 結論

参考文献

1. Vaswani et al., *Attention Is All You Need*, NeurIPS 2017. (arXiv)
2. Kaplan et al., *Scaling Laws for Neural Language Models*, 2020. (arXiv)
3. Hoffmann et al., *Training Compute-Optimal Large Language Models*, NeurIPS 2022. (arXiv)
4. Brown et al., *Language Models are Few-Shot Learners*, NeurIPS 2020. (arXiv)
5. Ouyang et al., *Training language models to follow instructions with human feedback (InstructGPT)*, NeurIPS 2022. (arXiv)
6. Schulman et al., *Proximal Policy Optimization Algorithms*, 2017. (arXiv)
7. Christiano et al., *Deep Reinforcement Learning from Human Preferences*, NeurIPS 2017. (arXiv)
8. Ziegler et al., *Fine-Tuning Language Models from Human Preferences*, 2019. (arXiv)
9. Stiennon et al., *Learning to summarize from human feedback*, NeurIPS 2020. (arXiv)
10. Rafailov et al., *Direct Preference Optimization (DPO)*, 2023. (arXiv)
11. Wei et al., *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*, 2022. (OpenReview)
12. Wang et al., *Self-Consistency Improves Chain of Thought Reasoning in Language Models*, 2022. (Astrophysics Data System)
13. Yao et al., *Tree of Thoughts: Deliberate Problem Solving with Large Language Models*, NeurIPS 2023. (OpenReview)
14. Yao et al., *ReAct: Synergizing Reasoning and Acting in Language Models*, ICLR 2023. (GitHub)
15. Schick et al., *Toolformer: Language Models Can Teach Themselves to Use Tools*, 2023. (arXiv)
16. Karpas et al., *MRKL Systems*, 2022. (arXiv)
17. Nakano et al., *WebGPT: Browser-assisted question-answering with human feedback*, 2021. (arXiv)
18. Yang et al., *SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering*, NeurIPS 2024. (arXiv)
19. Lewis et al., *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*, NeurIPS 2020. (arXiv)
20. Guu et al., *REALM: Retrieval-Augmented Language Model Pre-Training*, 2020. (arXiv)
21. Borgeaud et al., *Improving language models by retrieving from trillions of tokens (RETRO)*, ICML 2022. (arXiv)
22. Karpukhin et al., *Dense Passage Retrieval for Open-Domain QA*, EMNLP 2020. (arXiv)
23. Nogueira & Cho, *Passage Re-ranking with BERT (monoBERT)*, 2019. (arXiv)
24. Khattab & Zaharia, *ColBERT*, 2020. (arXiv)
25. Khattab et al., *DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines*, 2023. (arXiv)
26. Chen et al., *Evaluating Large Language Models Trained on Code (Codex)*, 2021. (arXiv)
27. GKV-developers, *gkvp: GyroKinetic Vlasov simulation code (repository)*. (GitHub)
28. Görler et al., *The global version of the gyrokinetic turbulence code GENE*, 2011. (MPG.PuRe)
29. Dorland et al., *Gyrokinetic Simulations of Tokamak Microturbulence* (GS2-related overview), 2000. (Princeton Plasma Physics Laboratory)
30. Holod et al., *Gyrokinetic particle simulations… (GTC-related)*, 2008. (AIP Publishing)
31. Nakata et al., *Validation studies of gyrokinetic ITG and TEM turbulence…*, *Nucl. Fusion* 2016. (NIFS Repository)
32. OpenHands, *OpenHands (coding agent platform, repository/site)*. (GitHub)
33. KyotoAI, *SEIMEI (repository)*. (GitHub)
34. KyotoAI, *RMSearch (repository)*. (GitHub)
35. DeepSeek, *DeepSeek-R1 (report/repository and coverage)*. (arXiv)