LLM推論・自己改善手法 詳細ガイド

性能を最大化する4つのアプローチと10の具体的アルゴリズム

1. Self-consistency

カテゴリ：Multiple Outputs & Voting（多数決）

概要：「3人寄れば文殊の知恵」をAI一人で実現する手法。1つの推論パスに依存せず、多様な推論を行い、その結果の多数決（アンサンブル）を取ることで正解率を高めます。

⚙ 動作メカニズム (Process Flow)

1. **多様なサンプリング**: 温度パラメータ(Temperature)を少し上げて、同じ質問に対して N 個の異なる回答生成を行う。
2. **推論パスの生成**: それぞれの生成で、異なる「思考の過程（CoT）」が出力される。
3. **整合性の集計**: 最終的な答え（例: "Answer is 5"）を集計し、最も頻度が高い（整合性が取れている）答えを採用する。

□ Merit
・CoTと組み合わせるだけで、数学や論理推論の正解率が劇的に向上する。
・追加の学習が不要で、推論時の工夫だけで実装可能。

⚠ Demerit
・回答を N 回生成するため、APIコストと計算時間が単純に N 倍になる。
・リアルタイム性が求められるチャットボットには不向き。

2. PREFER

カテゴリ：Multiple Outputs & Voting（プロンプト改善）

概要：回答ではなく「プロンプト（指示）」を進化させる手法。フィードバックループを回して、AI自身に「どう指示すれば解けるか」を考えさせ、最適なプロンプトを見つけ出します。

⚙ 動作メカニズム (Process Flow)

1. **Feedback (評価)**: 現在のプロンプトで問題を解かせ、どこが弱かったか、間違っていたかを特定する。
2. **Reflect (内省)**: なぜ失敗したのか？どのような情報や指示が不足していたのかを分析する。
3. **Refine (改良)**: 分析に基づき、より効果的な新しいプロンプトを生成する。
4. **Ensemble**: 改良された複数のプロンプトを用いて問題を解く。

□ Merit
・人間が試行錯誤（プロンプトエンジニアリング）する手間を自動化できる。
・タスク特有の「効く言い回し」をAIが発見してくれる。

⚠ Demerit
・「何をもって良いプロンプトとするか」の評価軸が必要。
・反復プロセスが長くなるとコストが増加する。

3. Self-improve

カテゴリ：Self-Feedback & Refinement（学習による自己強化）

概要：「自作の教科書で勉強する」アプローチ。ラベル付きデータがなくても、AI自身が生成した高信頼度の回答を正解データとして利用し、自分自身をファインチューニングします。

⚙️ 動作メカニズム (Process Flow)

1. **回答生成**: ラベルのない問題に対し、CoTとSelf-consistencyを用いて多数の回答を生成する。
2. **フィルタリング**: 生成された回答の中から、整合性が高く「自信がある（高信頼度）」ものだけを選別する。
3. **学習データの構築**: 選ばれた「問題＋推論＋答え」を教師データとする。
4. **Fine-tuning**: このデータを使ってモデル自体を再学習させ、基礎能力を底上げする。

□ Merit
・正解データが不足している専門領域でも精度向上が見込める。
・外部の人間によるアノテーションコストを削減できる。

⚠ Demerit
・誤った回答を高信頼度と判定してしまうと、嘘を学習するリスクがある。
・モデルの学習基盤が必要。

4. Self-refine

カテゴリ：Self-Feedback & Refinement（推論時の自己修正）

概要：「推敲（すいこう）」のプロセス。モデルの重みは更新せず、一度出した答えに対して自分でダメ出しを行い、修正版を作成することを繰り返します。

⚙ 動作メカニズム (Process Flow)

1. **Initial Output**: まず初稿（ドラフト）を作成する。
2. **Feedback**: 自身の出力に対し、「論理は正しいか？」「条件を満たしているか？」といった批評（Feedback）を生成する。
3. **Refine**: その批評をもとに、出力を修正（Revise）する。
4. **Loop**: 品質基準を満たすまで、このサイクルを数回繰り返す。

□ Merit
・追加学習なしで、プロンプトの工夫だけですぐに品質を向上できる。
・文章生成、コード生成など、正解が一つでないタスクに強い。

⚠ Demerit
・LLMの能力が低いと、的外れな批評をして逆に品質が下がる場合がある。
・トークン消費が増える。

5. Self-contrast

カテゴリ：Self-Feedback & Refinement（対比による再評価）

概要：「比較検討」のアプローチ。単に良し悪しを判定するのではなく、複数の案の「違い」に着目させることで、より客観的な見直しを行わせます。

⚙ 動作メカニズム (Process Flow)

1. **多様な視点の生成**: 異なる視点や推論パスで、複数の解決策（A案、B案）を生成する。
2. **差異の分析**: A案とB案を比較し、**「何が違うのか」**をリストアップさせる（採点はしない）。
3. **チェックリスト化**: その違い（例：計算手順の有無、観点の違い）を見直し用のチェックリストに変換する。
4. **再評価と修正**: チェックリストに基づいて、最終的な回答を洗練させる。

□ Merit
・LLMが苦手な「絶対評価（これは80点）」を避け、得意な「相対比較（AはBよりここが良い）」を利用できる。
・バイアスの軽減に役立つ。

⚠ Demerit
・違いは見つけられても、どちらが正しいかの判断を誤る可能性はある。

6. CRITIC

カテゴリ：Tool Feedback（外部検証）

概要：「Verify-then-Correct（検証してから修正）」フレームワーク。LLMの内部知識を過信せず、外部ツールを使ってファクトチェックを行います。

⚙ 動作メカニズム (Process Flow)

1. **回答生成**: まず通常通り回答を生成する。
2. **外部検証**: 回答内の事実や計算について、検索エンジン、電卓、コードインタプリタなどのツールを呼び出す。
3. **Feedback**: ツールの実行結果（検索結果や計算結果）をフィードバックとして受け取る。
4. **修正**: フィードバックに基づき、間違いがあれば回答を修正する。

□ Merit
・ハルシネーション（もっともらしい嘘）を劇的に減らせる。
・最新情報や正確な計算が必要なタスクに必須。

⚠ Demerit
・適切なツールを用意し、LLMに連携させるシステム設計が必要。
・外部APIの呼び出し待ち時間が発生する。

7. Voyager

カテゴリ：Tool Feedback（環境学習・スキル蓄積）

概要：Minecraft環境で実証された「生涯学習エージェント」。試行錯誤を通じて新しいスキル（コード形式の行動レシピ）を獲得し、ライブラリとして蓄積していきます。

⚙ 動作メカニズム (Process Flow)

1. **自動カリキュラム**: 自分の現在の能力に合わせて、次に挑戦するタスクを決める。
2. **反復的プロンプティング**: 行動コードを生成し、実行する。
3. **環境フィードバック**: 実行エラーやゲーム内の結果（死んだ、アイテム取れた）を受け取る。
4. **自己検証とスキル保存**: 成功したらそのコードを「スキル」としてデータベースに保存し、再利用可能にする。

□ Merit
・単発のタスク解決ではなく、経験を積み重ねてどんどん賢くなる。
・未知の環境でも自律的に適応できる。

⚠ Demerit
・フィードバックを返してくれるリッチな実行環境（ゲームやシミュレータ）が必要。

8. Reflexion

カテゴリ：Tool Feedback（言語的エピソード記憶）

概要：「失敗ノート」を作る手法。モデルの重みを更新する代わりに、失敗の原因と改善策を「言葉」にして短期記憶（コンテキスト）に保持します。

⚙ 動作メカニズム (Process Flow)

1. **実行 (Actor)**: タスクを実行し、環境からの評価を受け取る。
2. **評価 (Evaluator)**: 成功したか失敗したかを判定する。
3. **反省 (Self-Reflection)**: 失敗した場合、「なぜ失敗したか」「次はどうすべきか」を短い要約文（Reflection）にする。
4. **再挑戦**: 次の実行時に、そのReflectionをプロンプトに追加して、同じ失敗を避ける。

□ Merit
・強化学習のような重い計算なしで、試行錯誤による改善ループを回せる。
・人間が読んで理解できる形式で学習が進む。

⚠ Demerit
・コンテキストウィンドウ（入力可能な文字数）の制限を受ける。
・反省文が不適切だと、誤った方向に誘導される。

9. LLaMA-Rider

カテゴリ：Updating with Experience（成功体験の学習）

概要：「成功体験の刷り込み」。環境内での探索を通じて得られた「うまくいった軌跡」だけを集め、それを教師データとしてモデルを更新します。

⚙ 動作メカニズム (Process Flow)

1. **探索フェーズ**: 環境内でタスクを実行。フィードバックを受けて修正しながらゴールを目指す。
2. **フィルタリング**: 試行錯誤の中で、最終的に成功に至った一連の行動データだけを抽出する。
3. **学習フェーズ**: 成功データを用いて教師あり微調整（SFT）を行う。
4. **更新**: 次の探索では、更新された賢いモデルを使って、より難しいタスクに挑む。

□ Merit
・プロンプトに頼るだけでなく、モデル自体の「地頭」を良くできる。
・環境特有のルールを効率的に習得できる。

⚠ Demerit
・学習と探索を交互に行うパイプライン構築が必要で、コストが高い。

10. Chain of Hindsight (CoH)

カテゴリ：Updating with Experience（フィードバックの言語化学習）

概要：「後知恵（Hindsight）」を学習させる手法。単なる正解だけでなく、「良い例」と「悪い例」、そして「なぜ良い/悪いのか」というフィードバックそのものを学習させます。

⚙ 動作メカニズム (Process Flow)

1. **データ作成**: モデルの出力に対し、「この回答は良い（Positive）」「これは悪い（Negative）」という評価と理由を付与する。
2. **変換**: 評価を「良い回答の生成には...」「悪い回答を避けるには...」という自然言語のプロンプトに変換する。
3. **学習**: この「フィードバック付きデータ」でモデルを微調整する。
4. **推論**: 「役立つ回答をして」と指示するだけで、過去に学んだ「良い回答のパターン」が出せるようになる。

□ Merit
・RLHF（人間からのフィードバックによる強化学習）よりも手軽に、人間の好みを学習させられる。
・言語モデルの汎用性を維持しやすい。

⚠ Demerit
・質の高いフィードバックデータの準備が鍵となる。