高度な推論・行動設計：4つのアプローチ

LLMに「どう考えさせ、どう動かすか」の設計図

全体の見取り図：4つの手法の狙い

手法名	キーワード	狙い・特徴	得意な場面
DFSDT	**撤回とやり直し**	失敗したら分岐点まで戻る。 粘り強い探索。	APIエラーや検索失敗が 多いWebタスク
RadAgent	**スコアと直感**	過去の経験（Elo）から 「勝ち筋」を瞬時に選ぶ。	選択肢が多く、評価が 難しい意思決定
CoAct	**分業（上司と部下）**	計画するAIと実行するAIを 分けて、混乱を防ぐ。	手順が長いプロジェクト 出張計画・開発
Least-to-Most	**わらしべ長者**	簡単な小問から解き、 その答えを次のヒントにする。	難解な論理パズル 複雑な要因分析

1. DFSDT (Depth-First Search-based Decision Tree)

探索　バックトラック

🛠️ 何をする手法？

- ReAct（一本道）を拡張し、行動や推論を **「木（Tree）」** として分岐させる。
- 深さ優先で進み、ダメなら **「一つ前の分岐点に戻って（バックトラック）」** 別ルートを試す。

🚀 何が嬉しい？

- ツールやAPI呼び出しで失敗しても、その失敗を引きずらずに別手段へ切り替えられる。

> □ **具体例：レストラン予約**
> ルートA：食べログで探す → □予約不可（失敗）
> ルートB：公式サイトで探す → □フォーム発見（成功）
> Aがダメでも、諦めずに「その地点まで戻ってBを試す」が自然にできる。

2. RadAgent (Rational Decision-Making Agent)

効用関数　Eloレーティング

🛠️ 何をする手法？

- 「次の一手、どれが得か？」を判断するために、**Eloレーティング**（対戦ゲームの強さ指標）を使う。
- 「行動A vs 行動B」の比較を学習し、内在化されたスコア（直感）で自律的に判断する。

🚀 何が嬉しい？

- 外部からの正解ラベルがなくても、「どっちがマシか」の積み重ねで賢くなる。

> □ **具体例：安く買うタスク**
> 行動A：検索ワードを広げる vs 行動B：価格フィルタをかける
> 「Bの方が成功率が高い」と学習し、次からは自律的にBを優先するようになる。
> 玄人のような「勘（スコア）」を持つイメージ。

3. CoAct (Co-Action / Collaboration)

役割分担　階層的計画

🛠️ 何をする手法？

- **Global Planner（上司）**：全体の大枠計画とサブタスク定義を行う。
- **Local Executor（部下）**：具体的な環境操作・ツール利用を実行し、結果を返す。

🚀 何が嬉しい？

- 1つのLLMが全部やると混乱（コンテキスト汚染）しがちな長いタスクでも、役割を分けることで道筋が崩れにくい。

> □ **具体例：出張計画**
> **Global：**予算・日程から旅程案を作成。トラブル時は再計画。
> **Local：**「飛行機の空席確認」「ホテル予約」など個別の作業に集中。
> 「司令塔」と「現場」の連携プレー。

4. Least-to-Most Prompting

分解　段階的解決

🛠️ 何をする手法？

- 難しい問題を **「一番やさしいサブ問題から順番に」** 解いていく。
- **「前の問題の答え」** を次の問題の入力（ヒント）として使いながら、最終問題へ至る。

🚀 何が嬉しい？

- いきなり難問に挑むより、階段を作ることで推論精度が格段に上がる。

> □ **具体例：売上増の要因分析**
> Step 1（易）：まずは「一般的に売上が増える要因」を列挙。
> Step 2（中）：その視点で、対象企業のニュースを検索。
> Step 3（難）：集めた情報を整理し、根拠付きで結論を出す。
> 小問を積み上げて大問を解く「わらしべ長者」戦略。

設計・実装へのヒント：どの「型」を使う？

🧩 DFSDT風（分岐回復型）

- プロンプトに「失敗したら戻る」ルールを明記。LangGraph等でループ構造を作る。

⚖️ RadAgent風（比較評価型）

- 複数の案を出させ、それぞれの「成功率・コスト・リスク」を評価させてから実行する。

🏛️ CoAct風（組織型）

- Planner（計画役）とExecutor（実行役）でエージェントを分け、メッセージをやり取りさせる。

🪜 Least-to-Most風（階段型）

- プロンプトを複数回に分ける。「まず分解して」「Step1を解いて」「統合して」とチェーンさせる。