

SYRUP

# コンテキストエンジニアリング

海辺 康志

2026/09/19

# イントロダクション

- 前はLLMの内部構造を解説し、「**渡す文脈（コンテキスト）が変われば答えの質が変わる**」ことを理解した
- 今日はその設計法＝**コンテキストエンジニアリング**を理解する
- Claude CodeやCodexの「裏側で何が起きているか」
- **モデル（LLM本体）**と**ハーネス（周辺ツール）**は別物で、今日は主にハーネス側の話
- 2部構成で見てゆく
  - 1. 問題編**：コンテキストがなぜ有限か、どう壊れるか
  - 2. テクニック編**：Tool Use・ReAct・RAG・Compaction（圧縮）・サブエージェントなど、壊れを防ぐ対処法。
- 今日は「考え方」をみてゆく。次回のAGENTS.md / SKILLS / MCPの土台に

1. コンテキストエンジニアリングとは

2. コンテキストウィンドウとは何か

3. コンテキストの構成要素

4. コンテキストを扱う際の問題

5. コンテキスト管理の4操作

6. コンテキスト設計

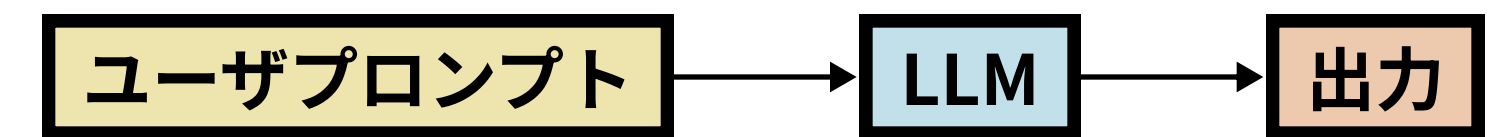
7. コンテキスト運用

8. チェックリスト

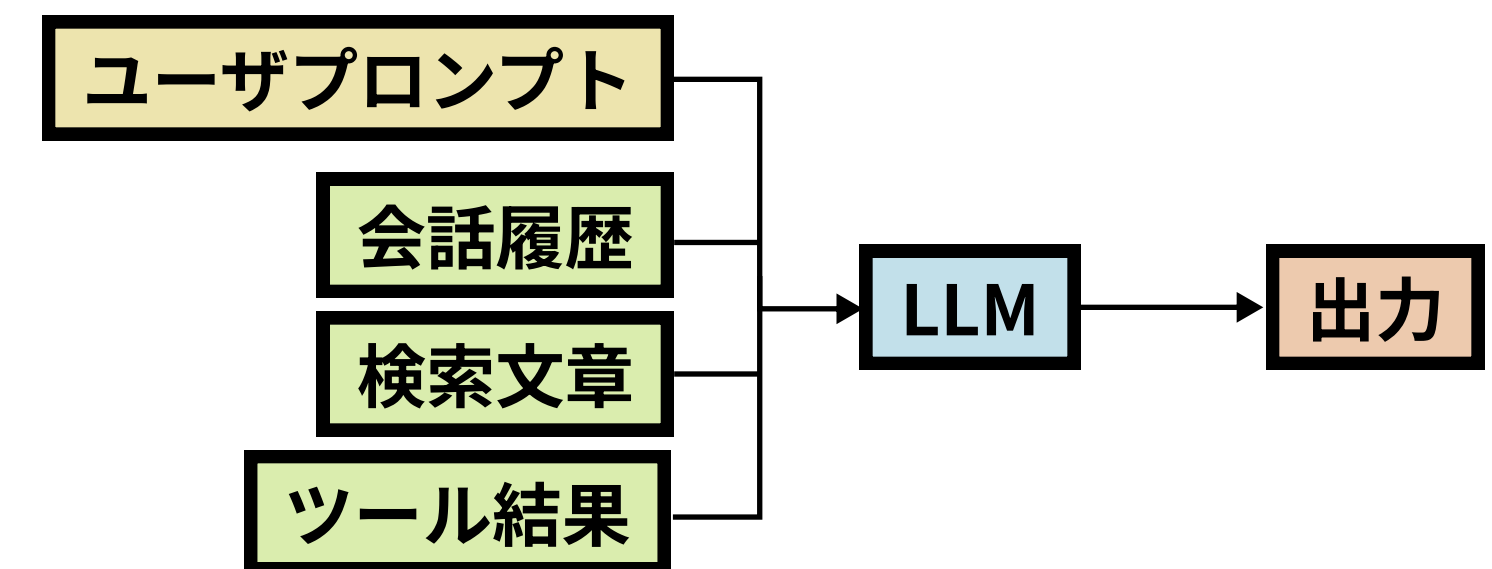
# コンテキストエンジニアリングとは

- 「**コンテキストエンジニアリング**」と似た言葉に「プロンプトエンジニアリング」があるが、違いを整理する
- **プロンプトエンジニアリング**：LLMへの指示文を工夫し、望む出力を得やすくすること
- しかしAIエージェントでは、指示文だけでなく**会話履歴・ツール結果・検索文書・メモリ**なども、回答に影響する
- **コンテキストエンジニアリング**：それらLLMに**渡す情報全体を、目的に合わせて選び・整理し続ける**こと
- 例えるなら、プロンプトエンジニアリングは「カーナビに出発前に目的地を正確に入力すること」、コンテキストエンジニアリングは「渋滞や事故などの最新情報をリアルタイムに反映しルートの最適化を続けること」

## プロンプトエンジニアリング



## コンテキストエンジニアリング



## 関連用語

コンテキストエンジニアリング

プロンプトエンジニアリング

# コンテキストとは

# コンテキストウィンドウ

- **コンテキストウィンドウ**：LLMが一度に読み込める情報量の上限（トークン数で表される）
- コンテキストが長くなるほど気をつけたいことは3つある
  1. **上限**：ウィンドウを超える情報は物理的に入らない
  2. **コスト・速度**：入力が高いほど処理負荷・料金・応答時間が増える（計算量はおよそ $n^2$ で増加）
  3. **実効的な使いにくさ**：上限内に収まっても、情報が多いほど必要な部分をモデルが正しく活用しにくくなることもある
- 仮に上限を20万トークンとすると、**原稿用紙で約500枚分、新書約2冊分**に相当する。それだけの情報が入っていても、必要な情報を確実に使いこなせるとは限らない

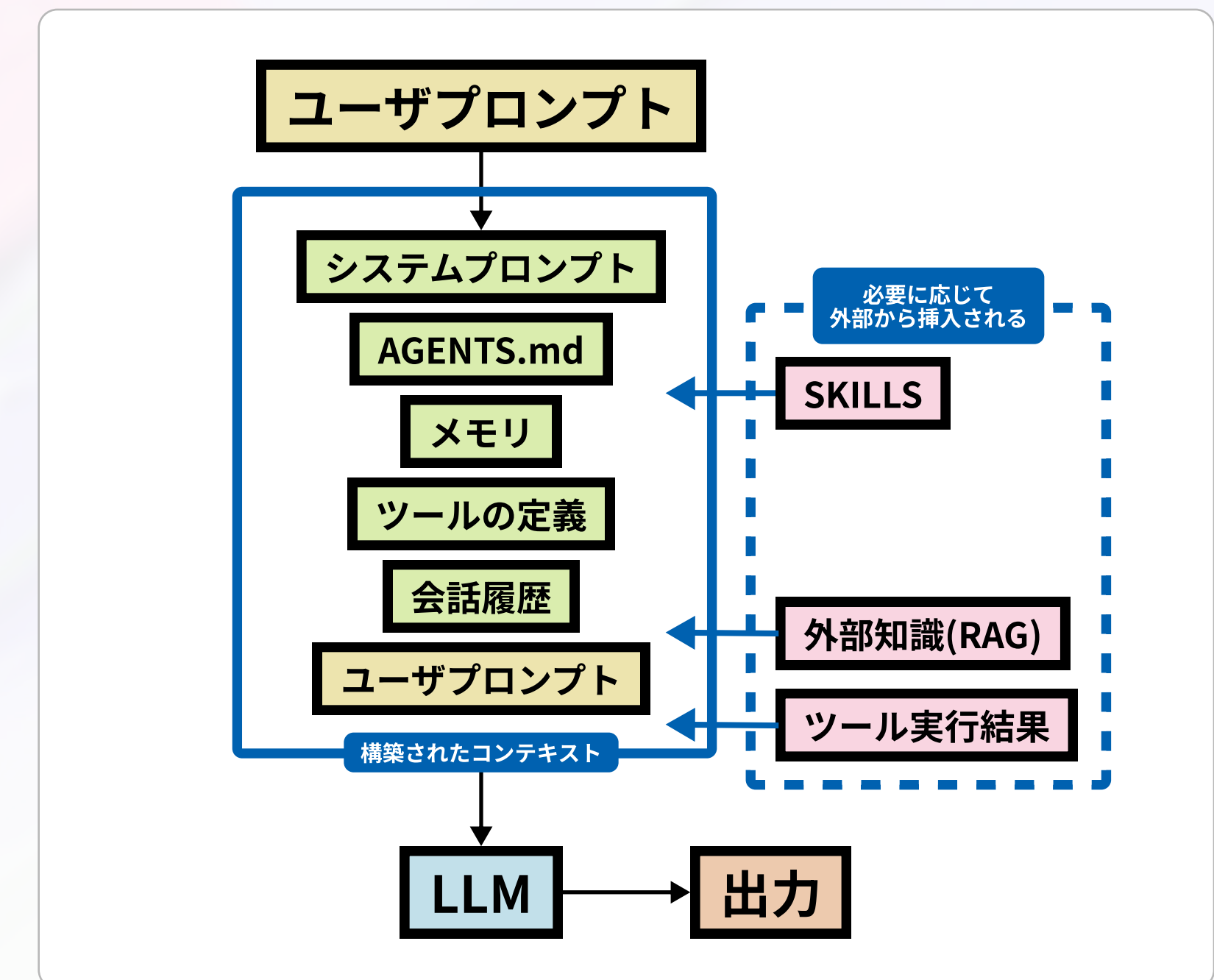


関連用語

コンテキストウィンドウ

# コンテキストの構成要素

- LLMに渡される「コンテキスト」は、複数の要素の組み合わせ
  - **システムプロンプト**：ハーネスが渡す役割・ルール・全体方針
  - **メモリ**：過去のセッションから引き継がれる情報
  - **ツールの定義・実行結果**：使えるツールの一覧や説明、および検索結果・コード実行結果など
  - **会話履歴**：これまでのやり取り
  - **外部知識（RAG）**：文書検索で取得した参考情報
- 注意点として「ツールを実行した結果」だけでなく「今どんなツールが使えるか」という定義自体も、使う・使わないに関わらず常にコンテキストを消費している
- どれか1つでも肥大化・劣化すると、コンテキスト全体の質が低下
- 「何を」「どの順番で」「どれだけ」渡すかが、コンテキストエンジニアリングの実質的な仕事



## 関連用語

システムプロンプト

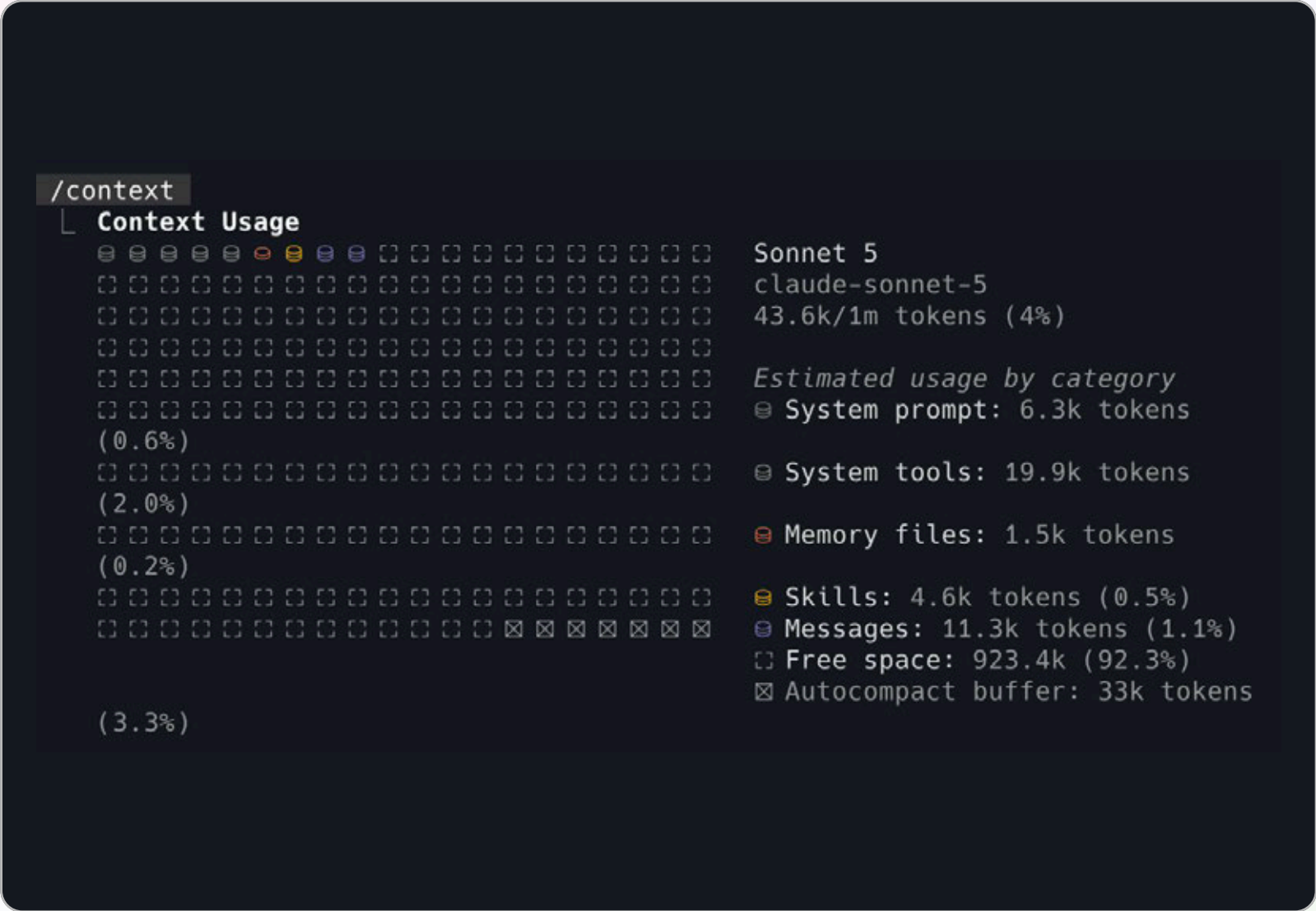
メモリ

会話履歴

RAG

# コンテキストの中身を覗く

- 実際に普段使っているツール上で中身を確認する方法がある
- Claude Code：`/context` コマンドでCLAUDE.md・MCPツール定義・スキルなど「何がコンテキストを消費しているか」の内訳を確認できる
- コンテキストの中身は「**固定費**」と「**変動費**」に分けて考えると理解しやすい
  - **固定費**：AGENTS.md、システムプロンプト、MCPツール定義など、毎ターン読み込まれる部分
  - **変動費**：会話履歴、読み込んだファイル、ツールの実行結果など、やり取りが進むほど増えていく部分
- 固定費が大きいほど、会話履歴・ファイル・ツール結果に使える枠は最初から狭くなる。



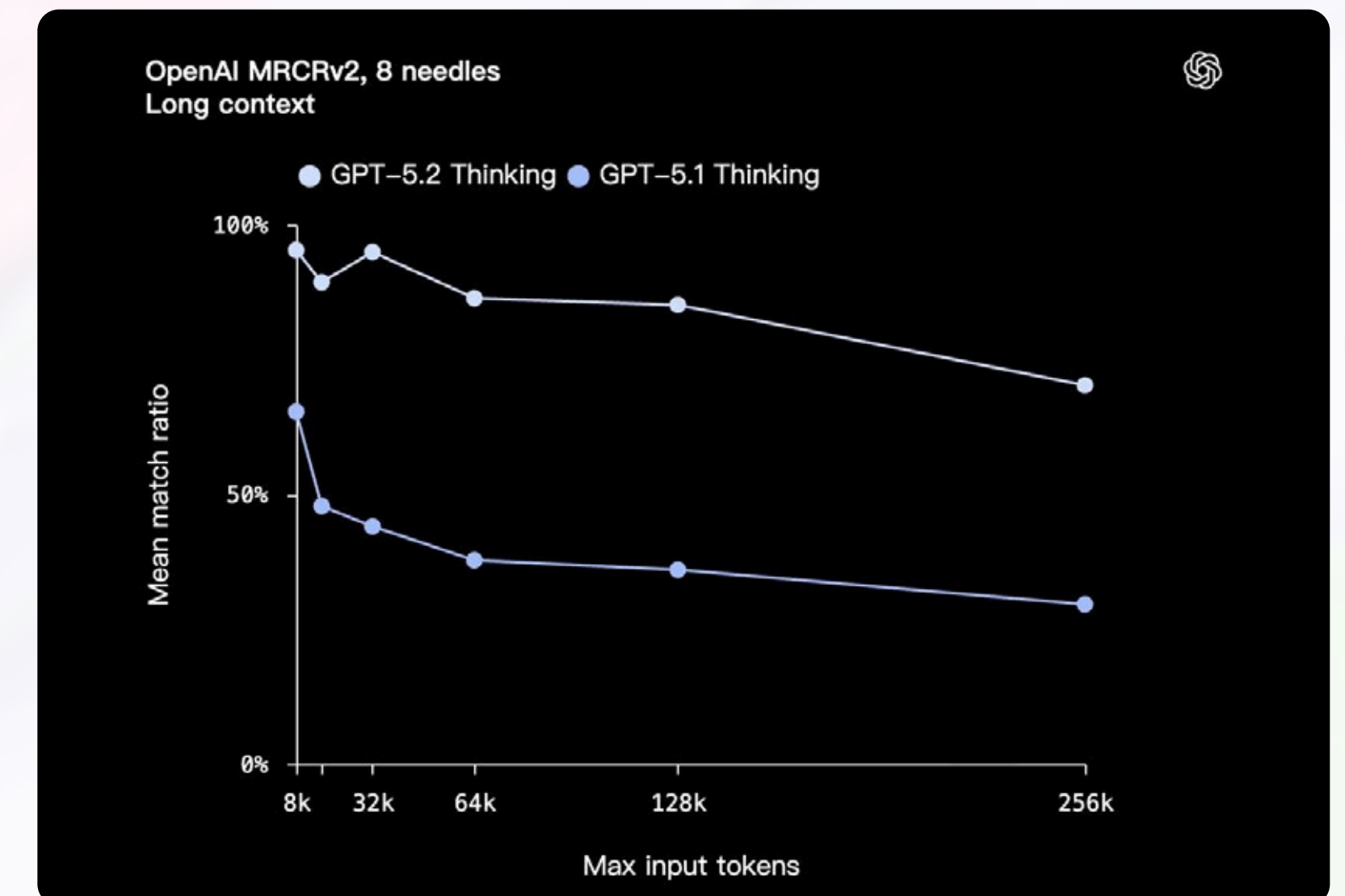
## 関連用語

contextコマンド

# コンテキストを扱う際の問題

## 問題①：コンテキストの劣化（Context Rot現象）

- 直感では「情報をたくさん詰め込むほど賢くなる」と思われがちだが、実は入力トークン数が増えるほど、モデルの精度が低下していく傾向がある
- これを**コンテキストの劣化（Context Rot）**と呼ぶ
- Chroma社が2025年に主要なLLMで検証：単純な「単語をそのまま繰り返すだけ」のタスクでさえ、入力が長くなると精度が落ちる傾向が多くモデルで見られた
- コンテキストは「大きければ大きいほど良い」道具ではなく、**使うほど目減りしていく有限な資源**として扱う必要がある



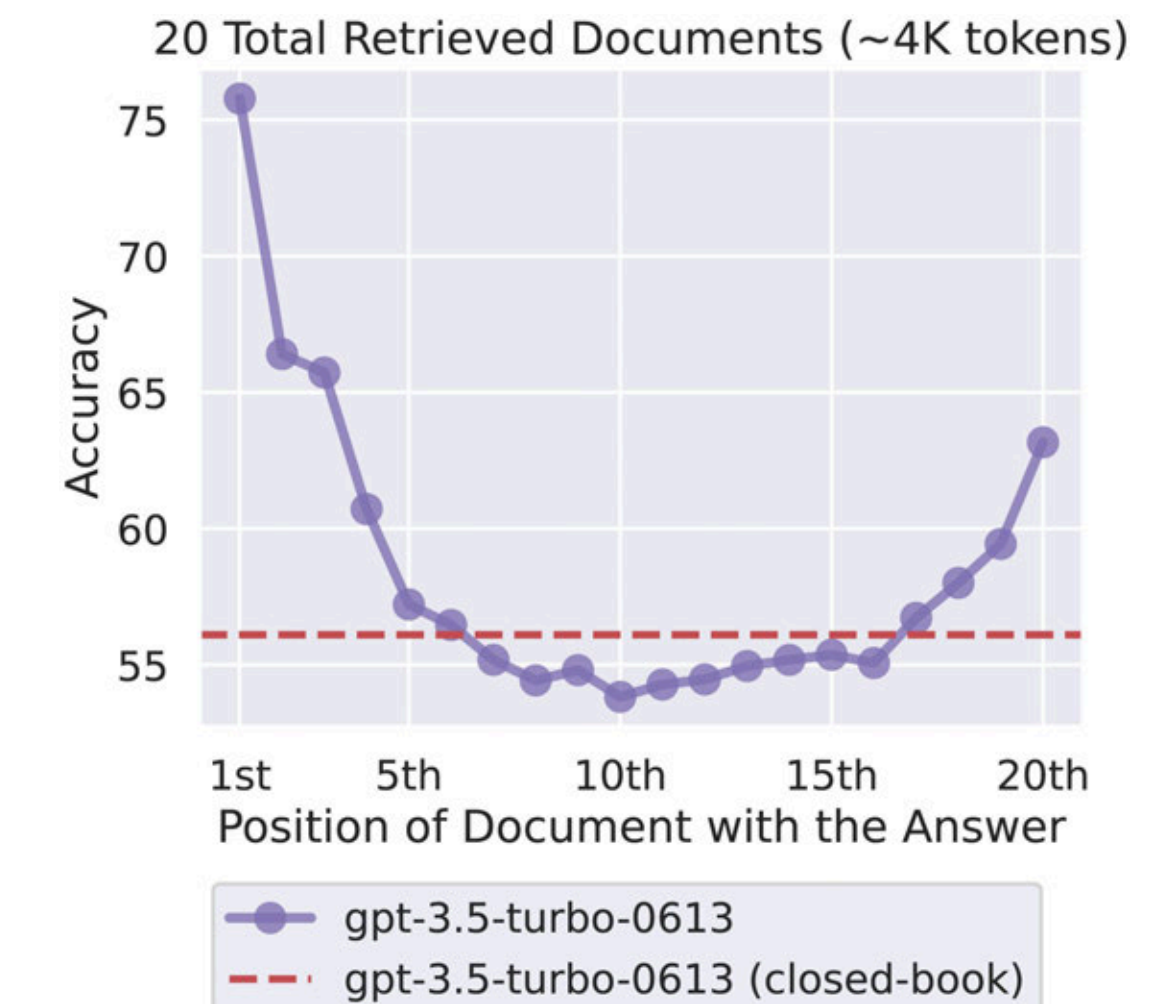
出典：OpenAI, “Introducing GPT-5.2,” 2025

### 関連用語

コンテキストの劣化

## 問題②：情報の位置による影響（Lost in the Middle）

- 特に有名な現象が**Lost in the Middle**で、これは**コンテキストの中間に置かれた情報ほど、見落とされやすい**という現象
- 精度をグラフにすると、**先頭と末尾が高く、中間が低いU字型カーブ**を描く傾向が報告されている
- スタンフォード大学の実験（2023年）：複数の文書の中から正解を探すタスクで、正解が中間付近に置かれると精度が目立って低下することが確認された
- 人間の記憶にも似た性質がある：最初（Primacy効果）と最後（Recency効果）はよく覚えているが、真ん中は忘れやすい



出典：Liu et al. “Lost in the Middle: How Language Models Use Long Contexts” (TACL 2024, arXiv:2307.03172)

### 関連用語

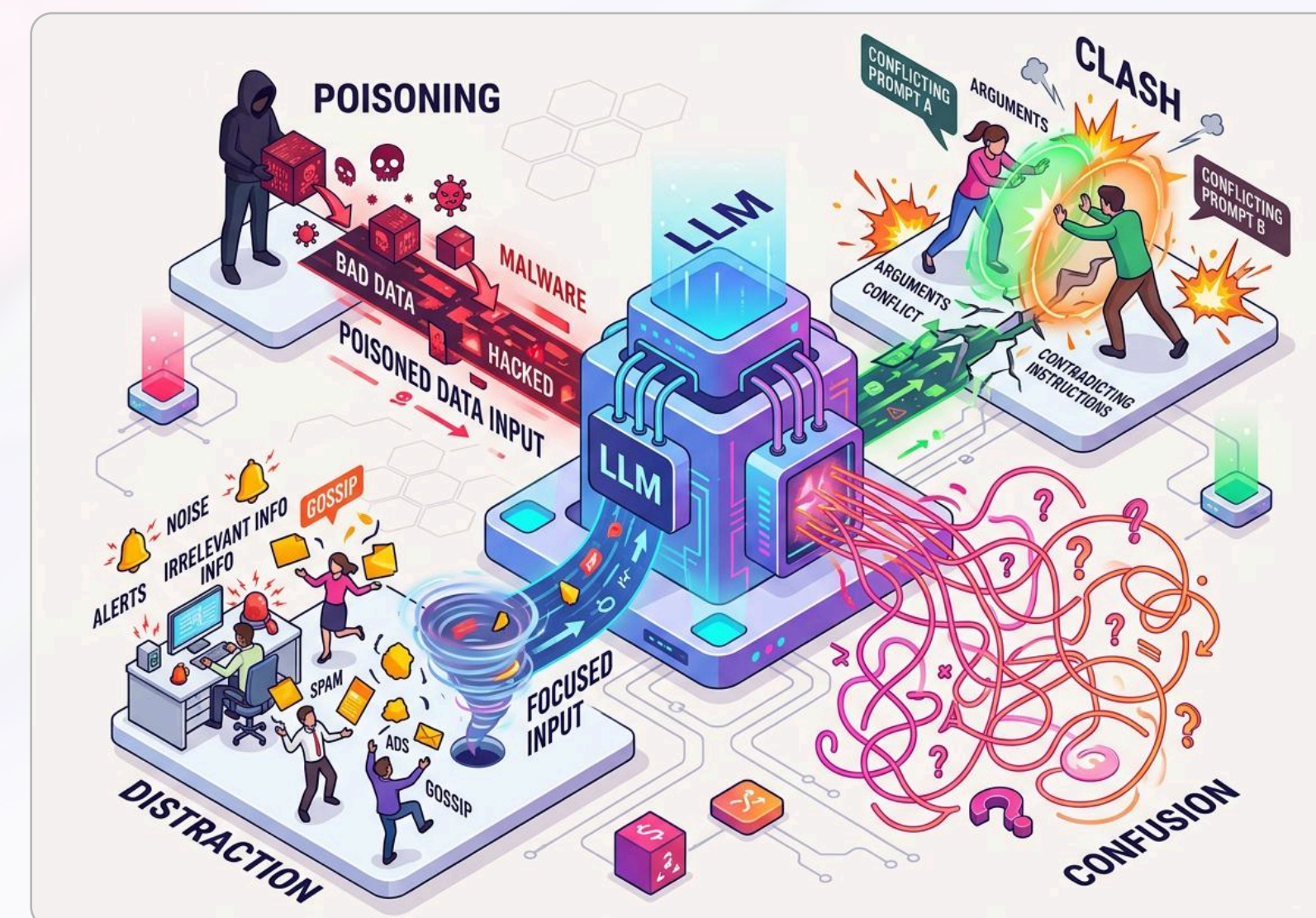
Lost in the Middle

Primacy効果

Recency効果

## 問題③：その他の落とし穴（Poisoning／Distraction／Confusion／Clash）

- その他にもコンテキストが壊れるパターンが知られている
- **Context Poisoning（汚染）**：一度紛れ込んだハルシネーションが、その後もずっと参照され続ける  
（例：存在しないファイル等を「あるはず」として扱い続ける）
- **Context Distraction（散漫）**：コンテキストが長くなりすぎて、モデルが目の前の文脈に引っ張られすぎる  
（例：少し前の重要な指示を急に無視し始める）
- **Context Confusion（混乱）**：不要な情報が混ざり、モデルが無理にそれを使おうとして精度が落ちる  
（例：関係のないツールを選ぼうとする）
- **Context Clash（衝突）**：コンテキストの中に矛盾する情報が混在し、モデルが混乱する  
（例：同じ修正とエラーを繰り返すループから抜け出せない）
- 外部から入った文章は、すべてを信頼しないようにする



### 関連用語

Context Poisoning

Context Distraction

Context Confusion

Context Clash

# コンテキスト設計

# プロンプトの小技

- 指示文を少し具体化するだけでも、回答の観点・形式・確かさを安定させやすい
  - 役割・観点を指定する**：「あなたはセキュリティの専門家です」のように、重視してほしい視点を伝える
  - 出力形式を指定する**：「結論→根拠→注意点の順」「表形式」のように、回答の形を決める
  - 制約を明示する**：「300字以内」「不明な点は推測せず、不明だと明記する」のように、守る条件を伝える
  - 確認行動を指定する**：「最新情報や不確かな点は、必ず検索して確認する」のように必要な行動を指示する
- これはモデルの能力向上の魔法ではなく、タスクに必要な観点・制約・手順を明示する工夫

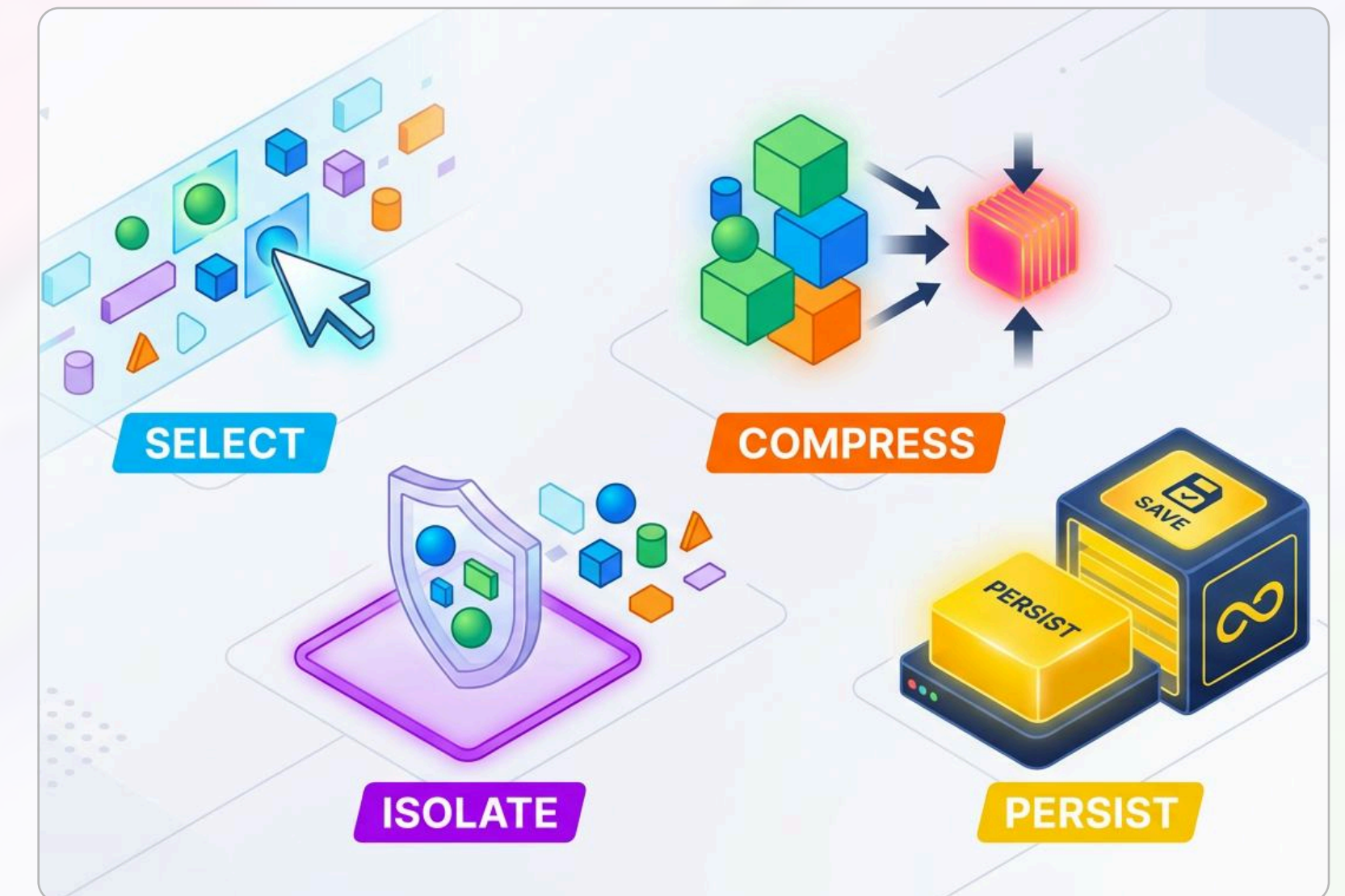


関連用語

ロールプロンプティング

# コンテキスト管理の4操作

- ここから先で紹介するテクニックは、実はすべて次の4つの操作のどれかに分類できる
  1. **選ぶ (Select)** : 必要な情報だけを取ってくる — Few-shot、RAG、Just-in-Time取得
  2. **圧縮する (Compress)** : 溜まった情報を要約して小さくする — Compaction
  3. **分離する (Isolate)** : 別のコンテキストに作業を切り出す — サブエージェント
  4. **外部に残す (Persist)** : セッションをまたいで永続化する — CLAUDE.md / AGENTS.md、メモリ
- 個別のテクニック名を覚えるより、「今やっているのはこの4つのどれか」で捉えたと理解・応用しやすい



## 関連用語

Select

Compress

Isolate

Persist

# コンテキスト設計①：Few-shot Prompting

- Few-shot Prompting：欲しい出力の「お手本」をいくつか例として一緒に渡すテクニック
- ゼロから説明するより、実例を2～3個見せる方が、フォーマットやトーンが伝わりやすく、出力も安定しやすい
- 例：文章等の書き方を統一したいとき、CLAUDE.md / AGENTS.mdに「良い例」を2～3個書いておくと、以降その形式に沿った出力が得られやすくなる
- ただし例を詰め込みすぎるとコンテキストを圧迫するため、**質の高い例を少数厳選する**のがコツ

以下の文章から情報を抽出し、JSONで出力してください。

例1

入力: 田中さんは東京在住、28歳です。

出力: {"name":"田中","age":28,"city":"東京"}

例2 入力: 鈴木さんは大阪在住、35歳です。

出力: {"name":"鈴木","age":35,"city":"大阪"}

入力: 佐藤さんは福岡在住、31歳です。

出力:

関連用語

Few-shot Prompting

## コンテキスト設計②：Function Calling / Tool Use

- **Function Calling (Tool Use)**：LLM自身に、外部の関数やAPI・ツールを「呼び出す」判断をさせる仕組み
- LLMはコードを実行せず、「このツールをこの引数で呼びたい」というリクエストを生成するだけ。実行は外側のプログラムが行い、結果をコンテキストに戻す
- 普段使うClaude CodeやCodexは、まさにこの仕組みの塊。ファイル操作・bash実行・Web検索は「LLMが呼び出しを要求→CLIが実行→結果を返す」の繰り返し
- **MCP (Model Context Protocol)**：AIと外部のツール・データ・定型指示をつなぐ共通規格。Figmaなど、対応しているツールではLLM側からそれを操作できるようになる



### 関連用語

Function Calling

Tool Use

MCP

## 深掘り：ツール設計のコツ

- ツールを渡せば渡すほど良いわけではない。Tool Use最大の落とし穴は「機能をカバーしすぎた、判断に迷うツール群」で、**似た機能のツールが増えるほどモデルは選択を誤りやすくなる**（Context Confusionの一種）
- **MCPサーバーの繋ぎすぎに注意**：使っていないサーバーを繋いでいるほどツール一覧が肥大化する。ツール名・引数・説明の定義は、**実行しなくてもコンテキストの固定費になる**
- 良いツール設計の原則：**役割が明確・機能が重複しない・説明が具体的**



関連用語

MCP

## コンテキスト設計③：ReAct（Reasoning + Acting）

- ReAct：「考える（Reasoning）」と「行動する（Acting）」を交互に繰り返す手法
- Thought（考える）→ Action（ツールを呼ぶ）→ Observation（結果を受け取る）→ また考えるの繰り返し
- Claude CodeやCodexの動作中の画面表示に近い
- 説明表示 → Bash/grep実行（Action）→ 結果表示（Observation）→ 次の判断、のループ
- 複数のタスク環境の実験で、成功率が大きく向上した例が報告されている
- 最近のモデルでは、「結論を出す前に、条件を整理して確認して」のように、**何を確認したいかを具体的に書く**と、自動でReActのサイクルが走ることが多い



### 関連用語

ReAct

## コンテキスト設計④：RAG（Retrieval-Augmented Generation）

- **RAG（検索拡張生成）**：質問に答える前に外部の文書・データベースを検索し、関連情報をコンテキストに追加する仕組み。ハルシネーション対策の代表例
- Claude CodeやCodexが「コードをファイルから検索し、関連部分だけをコンテキストに入れる」動きも広い意味でのRAG的発想
- プロジェクト全体を事前に読み込まず、必要な部分だけをその都度取得する。次に扱う「Just-in-Time取得」の一例
- **注意点**：取得した外部の文書・Webページ・ファイルの中身は信頼できるとは限らない。悪意ある指示文が紛れ込んでいると、LLMがそれをユーザー指示と誤認して実行してしまう**プロンプトインジェクション**のリスクがある



### 関連用語

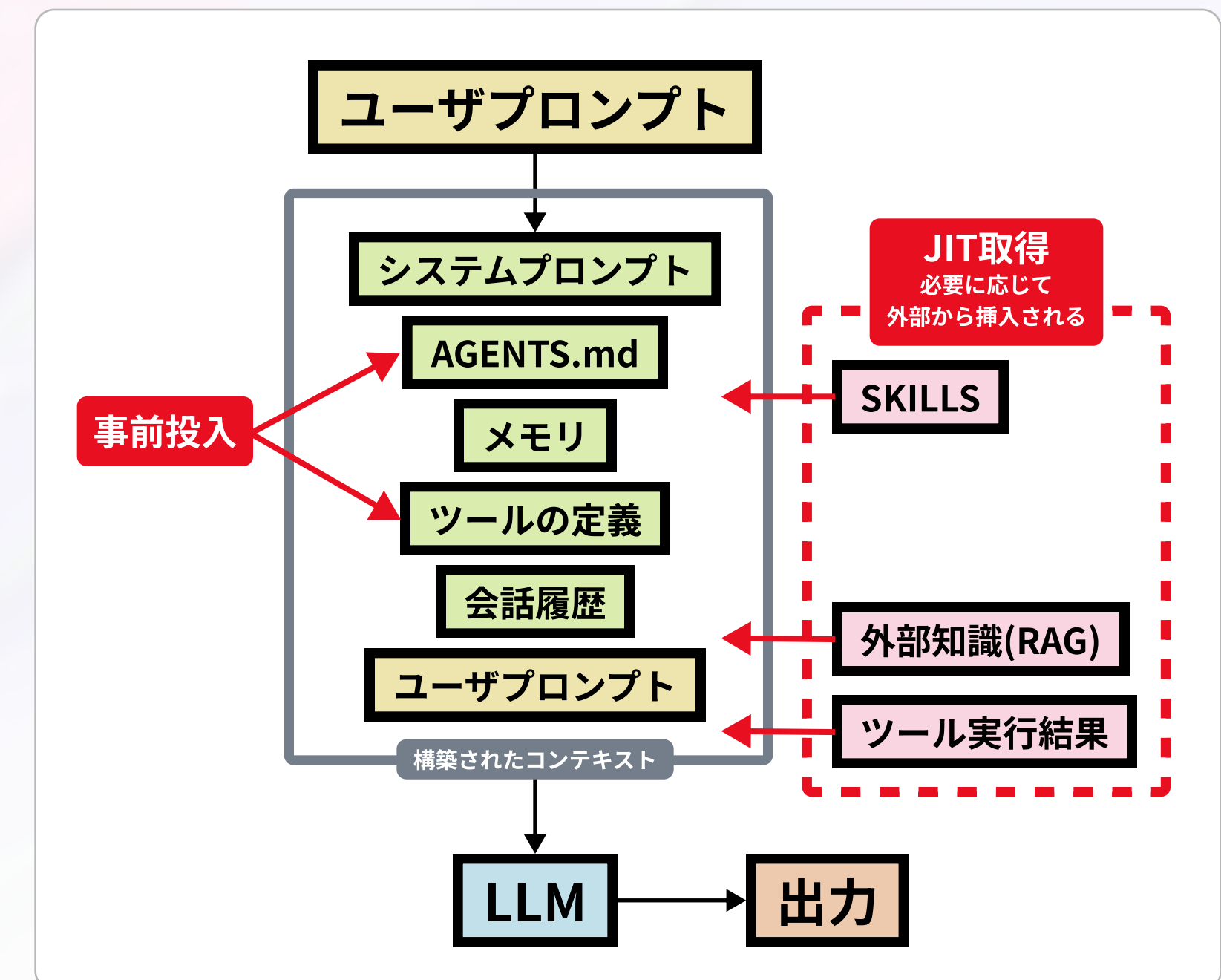
RAG

プロンプトインジェクション

# コンテキスト運用

## コンテキスト運用①：Just-in-Time取得 vs 事前投入

- 情報を渡すタイミングには、2つの設計がある
  - 事前投入 (Upfront Loading)**：毎回必要なルールを、最初からコンテキストに入れる
  - Just-in-Time (JIT) 取得**：必要になった時だけ、詳しい情報を取りに行く
- 事前投入の例：AGENTS.md / CLAUDE.md**
  - プロジェクト全体で常に守るルールを置く
  - ただし、長すぎると毎ターンの固定費になる
- JIT取得の例：SKILLS**
  - SKILLSは名前・説明を手がかりに、必要な場面だけで詳しい手順を読む。詳しくは次回に。



### 関連用語

Just-in-Time取得

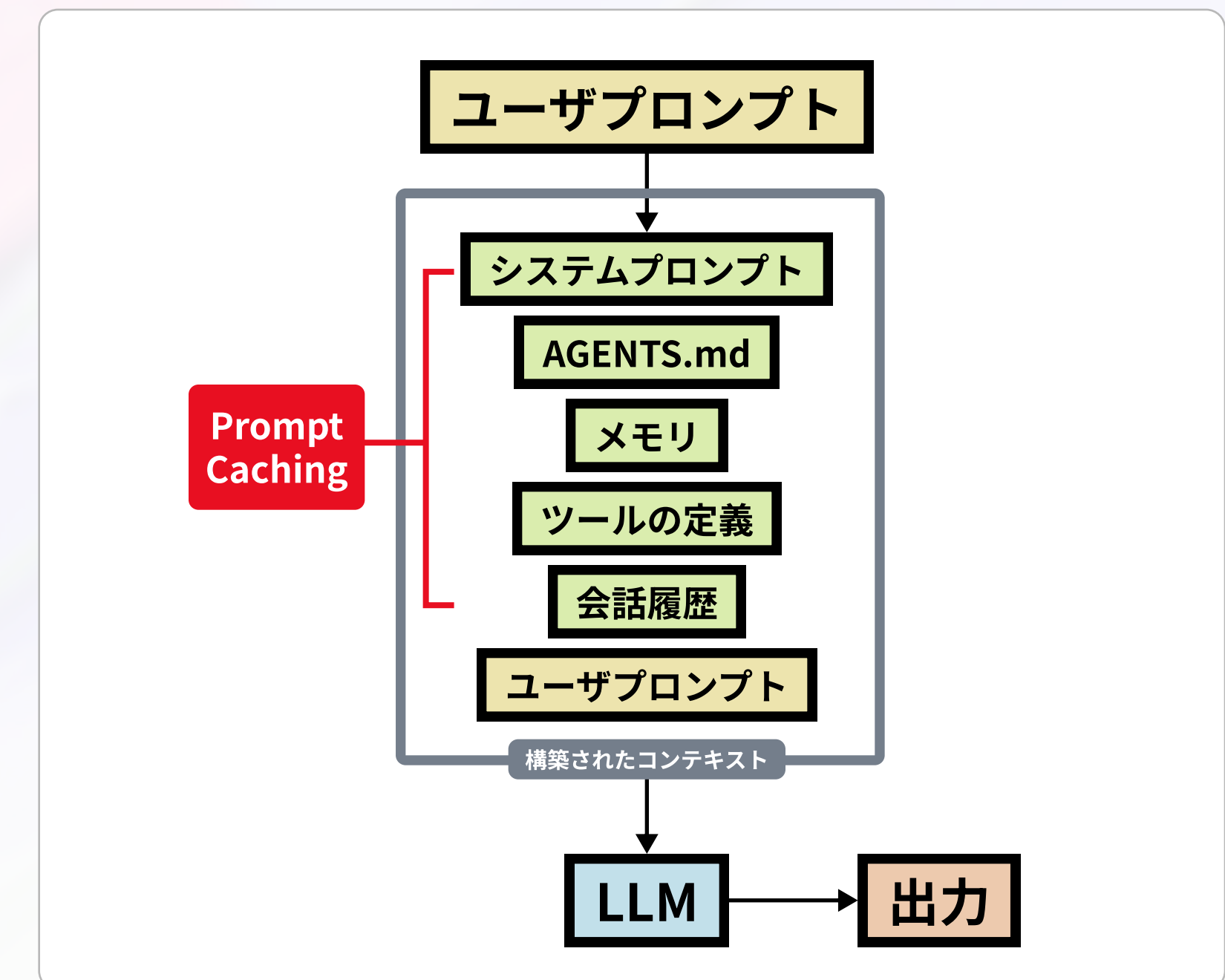
事前投入

事前投入

事前投入

## コンテキスト運用②：Prompt Caching（コスト・速度の最適化）

- **Prompt Caching（コンテキストキャッシュ）**：長いシステムプロンプトや事前投入された文章を一度処理した後、再利用することで**コストと応答時間を削減する仕組み**
- Claude CodeやCodexも裏側で活用しており、同じセッション内で会話を続けている間は再計算が省略される
- 「同じ前提を使い回すほど効率が良い」という背景を知っておくと、長時間の会話や頻繁なセッションリセットの使い分けの判断材料になる



関連用語

Prompt Caching

## コンテキスト運用③：圧縮・要約（Compaction）

- 長時間・長期間にわたる作業では、コンテキストウィンドウの上限にいずれ到達してしまう
- **Compaction（圧縮）**：これまでの会話・作業履歴を要約し、その要約から新しいコンテキストを作り直す手法
- **注意：圧縮は非可逆**。要約時に細部が失われたり、誤った前提がそのまま要約に固定されてしまうことがある。圧縮する前に「何を変更したか」「決定事項」「未解決事項」「次にやること」など明示的に書き出しておく安全
- Claude CodeやCodexでは、`/compact` で手動圧縮できる。上限が近づくと自動でも圧縮される



関連用語

Compaction

## コンテキスト運用④：サブエージェントへの委譲

- コンテキストが膨らみすぎる問題への、もう1つの解決策が「**作業をエージェント間で分担する**」こと
- **サブエージェント**：独立したエージェントを立ち上げ、そちらで詳細な調査を行った上で要点だけをメインの会話に返す仕組み。メインの会話が大量のログで汚染されない
- 向いているのは**独立して並列に調べられる幅広い探索**。密な情報共有が必要な一体型の実装には向かず、増やすほどトークン消費・統合の手間も増える
- Claude Codeは`**.claude/agents/**`、Codexは`**spawn\_agent**`と`**.codex/agents/**`でサブエージェントを定義・起動できる

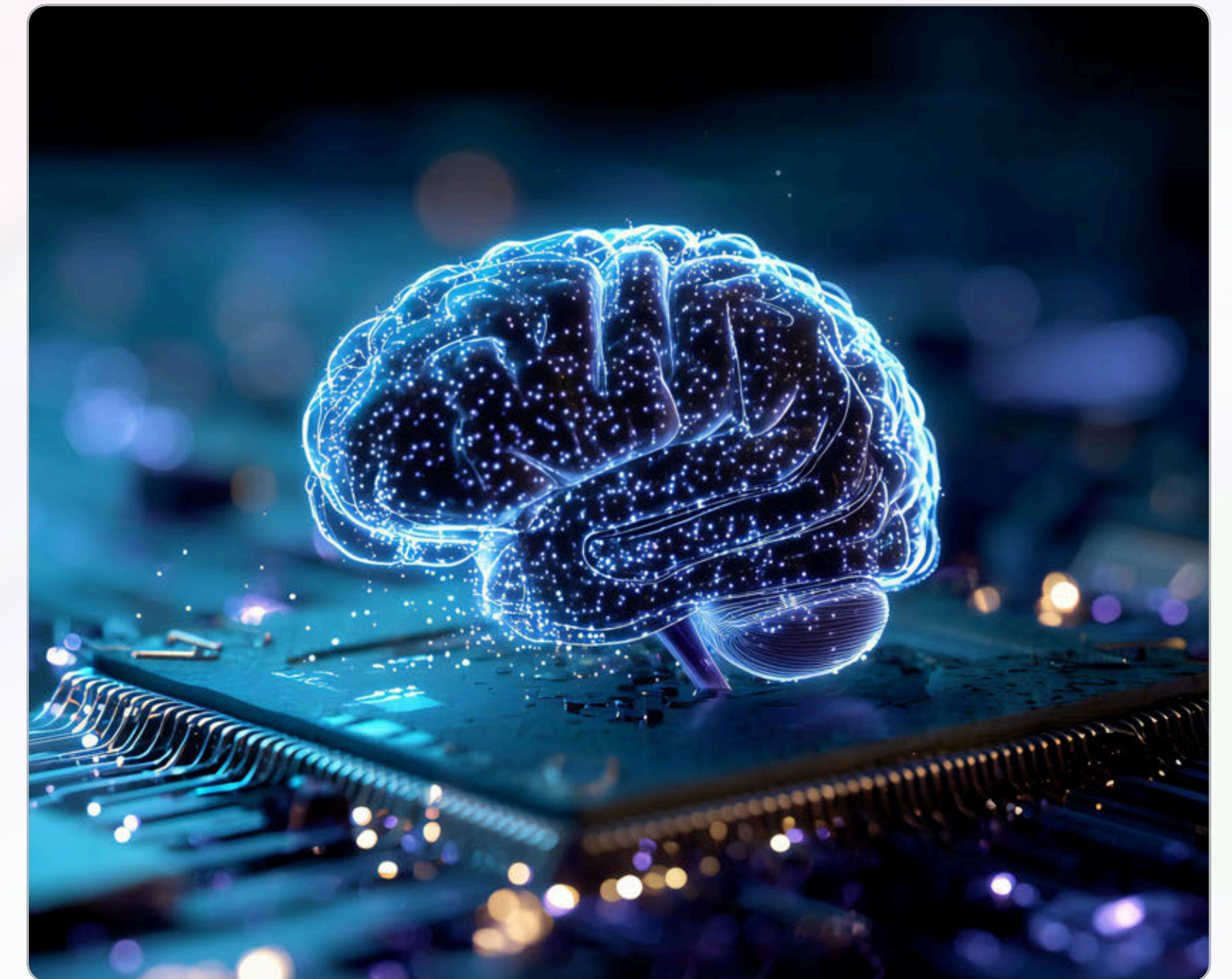


関連用語

サブエージェント

## コンテキスト運用⑤：永続コンテキスト

- ここまでのCompaction・サブエージェントは、あくまで「今のタスク中」だけの短期的な工夫
- それとは別に、セッションをまたいで情報を持ち越す**永続コンテキスト**という仕組みがある
- AIが自動的に記憶しているのではなく、**人間やハーネスがファイルに書いて残している**点がポイント
- **CLAUDE.md / AGENTS.md**がその代表例
- RAGとの違い：**RAGは必要になった時に検索して取りに行く**のに対し、**永続コンテキストはセッションが変わっても毎回同じ内容を最初から持っておく**
- 最近では、ハーネス自体に**AIが作業中に自ら学習内容を書き残す自動メモリ機能**が組み込まれはじめた



### 関連用語

永続コンテキスト

CLAUDE.md

AGENTS.md

# まとめ

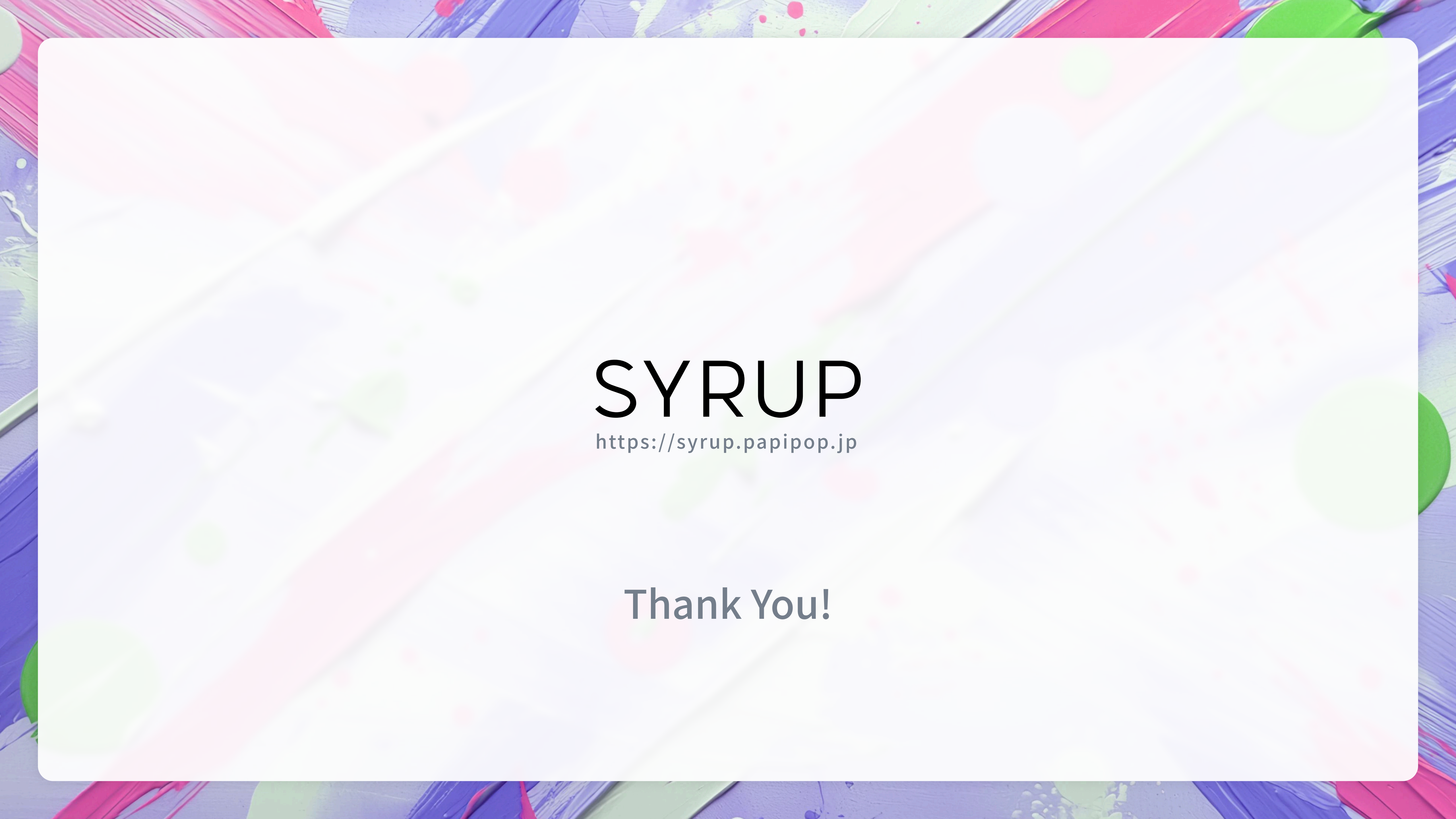
# 実践：コンテキストエンジニアリングのチェックリスト

- 今日の内容を、チェックリストとしてまとめる
  1. コンテキストは無駄に消費しない。使わないMCPサーバー等は繋がらない
  2. 指示文は具体的に：役割・出力形式・制約・確認行動を明示する
  3. 重要な指示は、CLAUDE.md / AGENTS.mdの先頭付近にまとめる（Lost in the Middle対策）
  4. 会話が長くなり様子がおかしくなったら、`/compact`などで一度整理したり、新規セッションでやり直す
  5. 大きすぎるタスクは、サブエージェントへの委譲を検討する
- Before / After 例
  - **Before（悪い例）**：「このフォルダ内を調べて、バグを直して」
  - **After（良い例）**：「ログイン後のプロフィール更新が失敗する原因を調査して。対象は`src/profile`に限定。まず原因と修正案を報告し、合意後に変更してください。完了時に変更ファイル・テスト結果・未解決事項をまとめてください」
  - **違いのポイント**：対象範囲を絞る／完了条件と報告形式を指定する／いきなり実装せず合意のステップを挟む



# コンテキストエンジニアリングの基礎は 理解できたでしょうか

気になるスライドがあったら、それを深掘りしてみてください。  
深掘りすることで、他の分野へと繋がり、世界がさらに広がってゆきます。



# SYRUP

<https://syrup.papipop.jp>

Thank You!