Anthropic.CCAR-F.v2026-08-22.q53

試験コード:	CCAR-F
試験名称:	Claude Certified Architect - Foundations
認定資格:	Anthropic
無料問題数:	53
バージョン:	v2026-08-22
アクセス数:	105
ページビュー数:	530
https://www.jpnpdf.com/Anthropic.CCAR-F.v2026-08-22.q53-mondaishu.html	

最新問題: **1**
Claude Agent SDK を使用して、開発者向け生産性向上ツールを構築しています。このエージェントは、エンジニアが馴染みのないコードベースを探索したり、レガシーシステムを理解したり、定型コードを生成したり、反復作業を自動化したりするのに役立ちます。組み込みツール（読み取り、書き込み、Bash、Grep、Glob）を使用し、Model Context Protocol（MCP）サーバーと統合します。
エンジニアが、既存のテストカバレッジが最小限である200個のファイルからなるレガシーコードベースに、包括的なテストを追加するようエージェントに依頼しました。エンジニアは、どのモジュールを優先すべきかを指定していません。エージェントはこの漠然としたタスクをどのように分解すべきでしょうか？
A. ディレクトリ構造に基づいて事前に固定のテストスケジュールを作成し、コードの複雑さやビジネス上の重要性に関係なく、各トップレベルディレクトリに均等な労力を割り当てます。
B. GlobとGrepを使用してコードベースの構造をマッピングし、密結合モジュールを特定し、影響の大きい領域に対して優先順位付けされた計画を作成し、依存関係が発見されたら修正します。
C. テストを作成する前に、200個のファイルすべてを体系的に読み、完全な機能一覧を作成し、テストを開始する前にテスト計画がすべての機能を網羅していることを確認します。
D. 最初のモジュールのテストをアルファベット順に書き始め、テストの失敗やインポートを利用して関連ファイルを自然に発見します。
Answer: B (メーッセージを残す**)**
重要なモジュールも必要なテスト手順も事前に分からないため、このタスクは未解決のままです。エージェントはまず、軽量な検出ツールを使用してリポジトリをマッピングし、既存のテストを特定し、中心となるモジュールを識別し、どのコンポーネントがファンイン性、ビジネス上の重要性、複雑な分岐構造、または広範な外部依存関係を持っているかを判断する必要があります。その後、リスクベースの初期テスト計画を作成し、新たな依存関係情報が現れたら計画を改良することができます。
Anthropicは、あらかじめ定義されたワークフローと、プロセスやツールの使用を動的に制御するエージェントを区別しています。エージェントは、必要な手順を確実にハードコーディングできず、環境証拠に適応する必要がある場合に適しています。実行中、エージェントはツールの結果を通じてグラウンドトゥルースを取得し、そのフィードバックを使用して後続のアクションを決定する必要があります。 https://www.anthropic.com/research/building-effective-

agents）Anthropicはまた、オーケストレーター -ワーカー設計は、必要なサブタスクが調査結果に依存する複雑なコーディングおよび検索タスクに適していると指摘しています。（https://www.anthropic.com/research）/効果的なエージェントの構築）
オプションAは、リスクではなくディレクトリ境界に基づいて作業量を割り当てます。オプションCは、価値を提供する前にコンテキストを徹底的に調査します。オプションDはアルファベット順を使用しますが、これは影響度やカバレッジの優先順位とは関係ありません。
オプションBは、証拠に基づいた分解手法を確立する。すなわち、影響力の大きい経路を発見し、優先順位を付け、テストし、結果を測定し、依存関係や未知のリスクが明らかになった場合に計画を修正する。
公式の参考文献／トピック：動的タスク分解、適応型エージェントループ、オーケストレーター -ワーカー、リスクベースのテスト計画。

最新問題: **2**
Claude Agent SDK を使用して、開発者の生産性向上ツールを構築しています。このエージェントは、エンジニアが馴染みのないコードベースを探索したり、レガシーシステムを理解したり、定型コードを生成したり、反復作業を自動化したりするのに役立ちます。組み込みツールである Read、Write、Bash、Grep、Glob を使用し、Model Context Protocol (MCP) サーバーと統合します。
エンジニアはレビューの際に、コードの変更内容とJiraチケットを照合するようエージェントに依頼することがよくあります。具体的には、チケットの説明、受け入れ基準、最近のコメントなどを確認します。現状では、これらの情報を会話に手動でコピー＆ペーストする必要があります。チームは、エージェントがこの標準的なJiraチケットデータに直接アクセスできるようにしたいと考えています。
最も効果的なアプローチは何でしょうか？
A. Bashツールとcurlを使用してJiraのREST APIを呼び出し、認証ヘッダーを含め、JSONレスポンスをインラインで解析します。
B. JiraのAPIをラップし、このチームのコードレビューワークフロー専用に設計されたツールを備えたカスタムMCPサーバーを構築する。
C. エージェントが読み取りツールを使用してアクセスするリポジトリに、Jira チケットを Markdown ファイルにエクスポートします。
D. 既存の Jira MCP サーバーを統合し、発見可能なツール インターフェースを通じてチケット、コメント、メタデータを公開します。
Answer: (解答を表示する**)**
オプションDは、不要なインフラストラクチャを作成することなく、既存の統合メカニズムを利用します。Anthropic社のClaude Code MCPドキュメントでは、ユーザーが課題トラッカーなどの外部システムから会話に情報を繰り返しコピーする場合に、MCPサーバーへの接続を特に推奨しています。接続が完了すると、サーバーは名前付きスキーマ定義ツールを通じてJiraの操作を公開し、Claudeはこれらのツールを直接検出して呼び出すことができます。これにより、エージェントはサーバーの認証とアクセス制御を維持しながら、チケットの説明、承認基準、コメント、メタデータを取得できます。
オプションAでは認証の詳細がシェルコマンドに公開され、エージェントがリクエストを構築してレスポンスを繰り返し解析する必要があります。オプションBは、既存のサーバーが必要な機能を提供していない場合、またはワークフローで高度に専門的な操作が必要な場合にのみ正当化されます。この質問では標準的なJira情報が必要なため、別のサーバーを

構築して維持する必要はありません。オプションCでは、権限、最新のコメント、および正式なチケットの状態が失われる、手動で同期された古いコピーが作成されます。したがって、チームは信頼できるJira MCP実装を選択し、可能な場合は適切な読み取り専用スコープを設定し、有効化する前にツールの説明とデータアクセス境界を確認する必要があります。

最新問題: **3**
あなたはClaude Codeを継続的インテグレーション／継続的デプロイメント（CI/CD）パイプラインに統合しようとしています。
このシステムは、自動コードレビューを実行し、テストケースを生成し、プルリクエストに対するフィードバックを提供します。実行可能なフィードバックを提供し、誤検出を最小限に抑えるようなプロンプトを設計する必要があります。
CIパイプラインに加えて、組織はこのリポジトリでClaude GitHubアプリを介してClaudeのマネージドコードレビューを有効にしており、すべてのプルリクエストでレビューが自動的に実行されます。レビューの平均
プルリクエストごとに18件の検出結果が報告されています。開発者からのフィードバックによると、不要なノイズは3つのカテゴリに分類されます。(1) CIのリンターで既に適用されているスタイルとフォーマットの問題、(2) src/gen/配下の自動生成されたテンプレートコードに関する検出結果、(3) 意図的なプロジェクト規約であるにもかかわらず、一般的なアンチパターンに似ているためにフラグが立てられるレンダリングヘルパーパターン。プルリクエストごとに、真のロジックバグはわずか4件程度です。
真の問題の検出能力を維持しながら、このノイズを低減するための最も効果的な方法は何でしょうか？
A. リポジトリのルートに REVIEW.md ファイルを作成し、CI で強制されるチェックと生成されたファイルのスキップルールと、レンダリング関連の検出結果が不正な動作を示す特定の行を引用するという検証要件を含めます。
B. GitHub Actions ワークフロー ファイルにカスタム レビュー手順を追加し、アクションの prompt パラメーターを使用して重複する lint 結果を抑制し、生成されたテンプレート コードを無視し、レンダリング関連の問題に対してより厳格な証拠要件を適用します。
C. プロジェクトの CLAUDE.md に、どのパターンが意図的なものか、リンティングは CI によって別途処理されること、src/gen/ ディレクトリには自動生成されたテンプレート コードが含まれていることなどを説明する詳細な説明を追加します。
Answer: (解答を表示する**)**
オプションAでは、管理対象のClaudeコードレビュー専用のコントロールサーフェスを使用します。Anthropicのコードレビューに関するドキュメントによると、ルートレベルのREVIEW.mdは、最優先の命令ブロックとしてすべてのレビューエージェントに挿入されます。このファイルでは、スキップパスの定義、CIによって既に適用されているカテゴリの抑制、重要度の再調整、ニット量の制限、特定の検出結果を報告する前にソースコードの証拠を要求するなどの設定が可能です。ドキュメントでは、生成されたコード、リンティング、検証要件が適切な使用例として明示的に示されています。
オプション B は、自己ホスト型の GitHub Actions ワークフローを構成しますが、このシナリオは Anthropic のインフラストラクチャ上で実行されている別の管理対象コードレビュー サービスに関するものです。このワークフローの手順は、管理対象のレビュー担当者を制御しません。オプション C は、有用な一般的なプロジェクト コンテキストを提供しますが、CLAUDE.md のレビュー固有の権限は低く、管理対象コードレビューでは、その違反は主に細かいレベルの指摘として扱われます。REVIEW.md は、管理対象サービスが報告する内容を変更するための、より強力で正確なメカニズムです。このファイルでは、src/gen/** をスキップし、CI によって既に強制されているスタイルの問題を抑制し、レンダリ

ング ヘルパーの警告には具体的な動作の証拠を要求する一方で、検証済みの正しさとセキュリティ上の欠陥については引き続き報告する必要があります。

最新問題: **4**
あなたはClaude Agent SDKを使用して、マルチエージェント型の研究システムを構築しています。コーディネーターエージェントは、専門的なサブエージェントに処理を委任します。サブエージェントは、Web検索、文書分析、調査結果の統合、レポート生成を担当します。このシステムは、様々なトピックを調査し、引用文献付きの包括的なレポートを作成します。
合成エージェントは最初の処理を完了しましたが、ウェブ検索エージェントと文書分析エージェントが特定のサブトピックに関する関連情報を見つけられなかったため、3つの重要な研究課題が未解決のままであると警告しました。コーディネーターは現在、レポート生成に直接進み、網羅性の低いレポートを作成しています。
研究の完全性を最も効果的に向上させるには、どのような変更が必要でしょうか？
A. ウェブ検索や文書分析に送信されるクエリの初期範囲を広げ、関連情報の見落としの可能性を減らします。
B. コーディネーターに合成結果のギャップを評価してもらい、その後、ターゲットを絞ったクエリを使用してウェブ検索と文書分析に再委任してから、再度合成を実行します。
C. レポート生成エージェントに、回答できなかった研究課題をメモさせ、ユーザーが最終出力の限界を理解できるようにする。
D. 合成エージェントにウェブ検索ツールへの直接アクセスを許可し、コーディネーターに制御を戻すことなく自律的に知識のギャップを埋められるようにする。
Answer: B (メーッセージを残す**)**
オプションBは、適切なオーケストレーション層で評価と改良のループを導入します。コーディネーターは既に研究計画と委任の決定を所有しているため、合成結果を必要な質問と照らし合わせて検査し、カバレッジのギャップを特定し、焦点を絞ったフォローアップの割り当てを発行する必要があります。Anthropicのマルチエージェント研究システムの説明はこのパターンに従います。リードエージェントは返された調査結果を合成し、追加の研究が必要かどうかを判断し、最終結果を生成する前に新しいサブエージェントを作成するか、戦略を改良します。オプションAのように初期クエリの幅を広げると、無関係な資料が追加で生成される可能性があり、予期しないギャップがカバーされることを保証できません。オプションCは、不完全性を修正するのではなく、単に不完全性を文書化するだけです。オプションDは、合成エージェントに検索機能を与えることで役割の分離を弱め、ツールの複雑さを増し、コーディネーターの中央集権的な追跡を迂回します。ターゲットを絞った再委任は、専門的な責任を維持し、研究、評価、改良、再合成の観察可能なシーケンスを作成します。コーディネーターは、明確な網羅基準を維持し、改良ラウンドの回数を制限することで、システムが制御不能な研究ループに陥ることなく、網羅性を向上させるようにする必要がある。

最新問題: **5**
自動コードレビューでは、プルリクエスト内の実際のバグが見落とされています。調査の結果、レビュープロンプトに「確実に本番環境の障害を引き起こす重大な問題のみをフラグ付けしてください」という指示が含まれていることが判明しました。
些細な懸念事項や不明な点は無視してください。開発者によると、見落とされた発見事項の中には、モデルが調査したものの報告しないことを選択した真の論理エラーが含まれているとのことです。チームは、レビュー出力が構造化され、すべての発見事項にメタデータがタグ付けされ、実行可能であることを求めています。抑制された発見事項の原因を取り除

き、下流のフィルタリングのために構造化されタグ付けされた出力を維持するには、どの変更をすぐに実行すればよいでしょうか？
A. 拡張思考を可能にし、モデルがレビューを生成する前に、すべてのコード変更について段階的に推論するように指示します。
B. モデルに、すべての結果を信頼度と深刻度タグ付きで報告するように指示し、フィルタリングは後続のステップに延期します。
C. 重症度に関する指示をすべて削除し、どの所見を報告するかについてモデルがデフォルトの判断を使用するようにします。
D. 同じプロンプトを使用して差分を再読み込みし、最初のレビューで見落とされた可能性のあるものを探す、2回目のレビュー パスを追加します。
Answer: B (メーッセージを残す**)**
この指示では、クロードに対し、不確実な結果や軽微な結果を抑制するよう明示的に指示しています。推論の深度を高めても、この報告ポリシーを覆すことはできません。クロードは分析中に実際の欠陥を特定したとしても、最終的な回答からそれを除外する可能性があります。オプションBでは、包括的な検出と受け入れフィルタリングを分離することで、自動的な意思決定に必要なメタデータを保持しつつ、偽陰性の原因を取り除きます。
Anthropicの現在のコードレビューに関するガイダンスでは、不確実な問題や深刻度の低い問題も含め、すべての問題を報告し、信頼度と推定深刻度を付与して、別の検証またはフィルタリング段階でランク付けすることを推奨しています。構造化されたスキーマでは、ファイルパス、行番号、カテゴリ、信頼度、深刻度、証拠、推奨されるアクションなどのフィールドを追加で要求することもできます。
オプションAは調査の質を向上させる可能性がありますが、抑制指示はそのまま残ります。オプションCは問題のある閾値を削除しますが、下流処理に必要な明示的な構造化分類も破棄します。オプションDは同じレビューポリシーを繰り返すため、2回目のパスでも1回目と同じ結果が抑制される可能性があります。オプションBは再現率と機械可読な出力を維持しながら、決定論的で調整可能な下流の閾値を可能にします。これは、モデルの初期検出タスク内に過度に制限的な受入決定を組み込むよりも信頼性が高いです。

最新問題: **6**
実際の運用環境では、「パリ協定が締結されたのは何年ですか？」といった単純な事実確認クエリでも、4つのサブエージェントすべてを順番に処理するため、クエリごとに40秒以上と大量のトークンを消費することがわかっています。複雑な比較調査では、パイプライン全体を利用することでメリットが得られます。ユーザーが新しいアプリケーションを発見するにつれて、クエリの分布は多様化し、変化しています。クエリの複雑さが変化する状況で最適化するための最も効果的なアプローチは何でしょうか？
A. 事実に関する質問に対しては、サブエージェントを完全にバイパスする高速パスを作成し、その他のすべてのクエリは完全なパイプラインを通してルーティングすることで、調査の徹底性を確保します。
B. ラベル付けされた過去のデータに対してクエリ複雑度分類器を訓練し、最適なサブエージェントの組み合わせを予測します。クエリパターンが変化するにつれて、定期的に再訓練します。
C. コーディネーターが各クエリを分析し、クエリ要件の評価に基づいてどのサブエージェントを呼び出すかを動的に決定します。
D. 構造（単一事実比較、分析）に基づいてクエリを分類し、各カテゴリを事前定義されたサブエージェントの組み合わせにマッピングするパターンベースのルーティングを実装します。

Answer: C (メーッセージを残す**)**
オプションCでは、各リクエストの実際の証拠と処理要件に応じて、調整作業の規模を調整できます。事実確認であれば、1回の検索操作とコーディネーターによる直接的な対応で済むかもしれませんが、比較調査であれば、並行検索、文書分析、統合、正式な報告書の作成が必要となる場合があります。
Anthropicのオーケストレーター・ワーカー型ガイダンスでは、タスクを動的に分解し、ワーカーを選択し、その結果を組み合わせる中央モデルについて説明しています。このパターンは、必要なサブタスクを事前に確実に予測できない場合に特に適しています。Anthropicのマルチエージェント研究アーキテクチャも同様に、リードエージェントを使用してクエリを分析し、適切な研究戦略を決定します。
オプションAは柔軟性に欠ける2経路システムを構築し、事実に基づかないすべてのクエリを依然として最もコストのかかるワークフローに通します。オプションBは、ラベル付きトレーニングデータ、外部で管理される分類器、および最適なエージェントの組み合わせの安定した定義を必要とします。オプションDは実装コストは低いものの、クエリの種類が事前に定義されたカテゴリを超えて進化すると脆弱になります。動的調整では、モデルの既存の推論能力を使用して、ソース要件、分析深度、期待される出力、および不確実性を評価し、結果を大幅に改善する専門家のみを呼び出します。

最新問題**: 7**
あなたはClaudeを使用して構造化データ抽出システムを構築しています。このシステムは、非構造化ドキュメントから情報を抽出し、JavaScript Object Notation（JSON）スキーマを使用して出力を検証し、高い精度を維持します。また、エッジケースを適切に処理し、下流システムと統合する必要があります。
抽出パイプラインは出力結果をJSONスキーマと照合して検証しますが、レビュー担当者の能力が限られているため（レビュー担当者は抽出総量の約5%しか処理できません）、人間のレビューを実装する必要があります。
人間の審査に回す抽出データを選択するための最も効果的な基準は何ですか？
A. モデルが信頼度が低いと示している場合、またはソース文書に曖昧または矛盾した情報が含まれている場合の経路抽出。
B. 抽出の信頼性に関係なく、特定の優先度の高いエンティティタイプ（例：財務数値、日付）を含む抽出を人間のレビュー対象とする。
C. 下流システムがデータ品質の問題または処理の失敗を報告した場合にのみ、ルート抽出をレビューします。
D. 抽出液の5%を無作為に抽出してレビューする。
Answer: (解答を表示する**)**
限られたレビュー能力は、意味的な誤りが発生する可能性が最も高い記録に集中させるべきである。
スキーマ検証は、応答が正しい構造とデータ型を持っていることを確認するものであり、抽出された値が正確であることを保証するものではありません。曖昧な表現、矛盾する記述、証拠の欠落、モデルで報告された不確実性などは、抽出に人間の判断が必要であることを示す直接的な指標です。
Anthropic の信頼性に関するガイダンスでは、幻覚低減法ではエラーを完全に排除できないため、Claude が不確実性を表明できるようにし、事実に基づいた出力をソース資料に基づかせ、重要な情報を検証することを推奨しています。これらの原則は、構文的に有効な出力を自動的に信頼できるものとして扱うのではなく、不確実または証拠的に矛盾する記録をレビュー担当者に送ることを支持します。(https://docs.anthropic.
(com/en/docs/test-and-evaluate/strengthen-guardrails/reduce-hallucinations) オプション B では、明確で十分に裏付けられた値にレビュー担当者のリソースを浪費する一方で、他のフィールドの曖昧なエラーを見落とす可能性があります。オプション C はリアクティブです。下流での承認は意味の正しさを保証するものではなく、サイレントエラーは処理エ

ラーを引き起こさない可能性があります。オプション D は全体的な品質の偏りのない推定値を提供するため、二次監査メカニズムとして保持する必要がありますが、5% しかレビューできない場合、ランダム選択はリスクのあるレコードを検出する最も効率的な方法ではありません。
実運用においては、モデルの信頼度をそれ自体で較正された確率として扱うべきではありません。矛盾する情報源の範囲、欠落している引用、OCRの品質、検証警告、文書の種類、ビジネスへの影響といった客観的なシグナルと組み合わせて使用する必要があります。
公式参照資料／トピック :ヒューマン -イン -ザ -ループ -レビュー、不確実性処理、ソース -グラウンディング、リスクベースのエスカレーション。

最新問題: **8**
あなたはClaude Agent SDKを使用して、マルチエージェント型の研究システムを構築しています。コーディネーターエージェントは、専門的なサブエージェントに処理を委任します。サブエージェントは、Web検索、文書分析、調査結果の統合、レポート生成を担当します。このシステムは、様々なトピックを調査し、引用文献付きの包括的なレポートを作成します。
合成エージェントは、ウェブ検索エージェントと文書分析エージェントから要約された結果を受け取り、統合された要約をレポート生成エージェントに渡します。テスト中に、生成されたレポートに適切な引用のない事実主張が含まれていることが判明しました。要約処理中にメタデータが失われたため、レポート生成エージェントは記述を元の情報源に紐付けることができません。
最終報告書において、適切な情報源の明記を確実に行うための最も効果的なアプローチは何ですか？
A. 各エージェントが、コンテンツの概要と、URL、ドキュメント名、ページ番号などのソースメタデータを分離した構造化データを出力します。
B. 要約をスキップし、Web検索と文書分析からの完全な生の出力をレポート生成器に直接渡します。
C. 合成エージェントに、一貫した引用形式を使用して、要約テキスト内にソース参照をインラインで埋め込むように指示します。
D. レポート生成ツールがウェブ検索エージェントにクエリを実行して、最終レポート内の主張の出典を特定するようにします。
Answer: (解答を表示する**)**
オプション A では、パイプライン全体を通して出所情報が第一級データとして保持されます。各発見には、要約された主張、裏付けとなる抜粋、ソース識別子、URL または文書タイトル、およびページやコンテンツブロックインデックスなどの位置情報が含まれる必要があります。合成エージェントは、証拠とソースの関連付けを切断することなく発見を組み合わせることができ、レポートジェネレーターは構造化されたレコードから直接引用を生成できます。Anthropic の引用に関するドキュメントでは、信頼できる引用では、裏付けとなる箇所とその正確なソース位置 (PDF ページ範囲やコンテンツブロックインデックスを含む) が特定されると説明されています。すべての生出力を渡すオプション B では、証拠は保持されますが、過剰なコンテキストが導入され、ソースの選択が難しくなります。
選択肢Cのように、文章中のインライン参照は、その後の要約時に変更、省略、または主張から切り離される可能性があります。選択肢Dは、レポート作成後に調査を繰り返すため、当初使用した証拠とは異なる情報源を見つける可能性があります。したがって、構造化された出所契約は、すべてのサブエージェントの出力に必須であり、各引き渡し時に検証されるべきです。レポート作成者は、関連する証拠記録がない事実の主張については、引用を捏造したり、後から再構築したりするのではなく、拒否またはフラグを立てるべきです。

最新問題: **9**
あなたはClaude Codeを継続的インテグレーション／継続的デプロイメント（CI/CD）パイプラインに統合しようとしています。
このシステムは、自動コードレビューを実行し、テストケースを生成し、プルリクエストに対するフィードバックを提供します。実行可能なフィードバックを提供し、誤検出を最小限に抑えるようなプロンプトを設計する必要があります。
自動コードレビューを導入した後、開発者たちは、検出された問題の約35%が誤検出であり、そのパターンは一貫していると報告している。具体的には、チームの慣習に反するスタイルの提案、デプロイ環境では安全なパターンに対するセキュリティ警告、そしてこの特定のユースケースのパフォーマンスを低下させるようなパフォーマンスに関する提案などである。
偽陽性を減らしつつ、モデルがこれまで見たことのない新しいコードパターンに対しても判断を一般化できるようにしたい。
どの方法が最も効果的ですか？
A. フラグを立ててはならないすべてのパターンの包括的な仕様を作成し、システムプロンプトに完全なドキュメントを含めます。
B. 各カテゴリにおいて、許容されるプロジェクトパターンと実際の問題点を区別する注釈付きコードスニペットを含む、少数の例を含める。
C. キーワードベースの後処理を使用して、「convention」などの用語を含む検索結果を削除します。
「状況依存的」あるいは「トレードオフ」。
D. クロードに慎重な対応をし、明確な問題のみを報告するようにという一般的な指示を追加します。
Answer: B (メーッセージを残す**)**
オプションBは、クロードが習得すべき判断基準を示しています。厳選された例を用いることで、構造的に類似したコードでもプロジェクトの状況によって異なる結果が生じることを示すことができます。例えば、承認済みの認証ラッパーと安全でない直接呼び出し、あるいは意図的なパフォーマンスのトレードオフと偶発的な二次演算などが挙げられます。こうした対比を通して、クロードは禁止語句を単に暗記するのではなく、新しいコードにもその根底にある判断基準を適用できるようになります。
Anthropicは、出力の精度と一貫性を向上させる最も信頼できる方法の1つとして、関連性があり、多様で、明確に構造化された例を挙げています。実際のユースケースを反映し、重要なエッジ条件を網羅する複数の例を推奨しています。オプションAは、コンテキストを消費し、維持管理が困難になり、将来のあらゆるバリエーションを予測できない、過度に大きなネガティブカタログを作成するリスクがあります。オプションCは生成後にテキストをフィルタリングするため、選択されたキーワードを使用している真の発見を抑制してしまう可能性があります。オプションDは危険なほど曖昧です。レビュー担当者に保守的であるように指示すると、実際だが不確実な欠陥を抑制し、リコール率を低下させる可能性があります。プロンプトでは、許容可能な例と問題のある例をペアで提示し、各分類が異なる理由を説明し、報告されたすべての発見について具体的なコード証拠を要求する必要があります。Anthropicプロンプトのベストプラクティス

最新問題: **10**
Claude Agent SDK を使用して、開発者向け生産性向上ツールを構築しています。このエージェントは、エンジニアが馴染みのないコードベースを探索したり、レガシーシステムを理解したり、定型コードを生成したり、反復作業を自動化し

たりするのに役立ちます。組み込みツール（読み取り、書き込み、Bash、Grep、Glob）を使用し、Model Context Protocol（MCP）サーバーと統合します。
コードベース探索ツールはセッションIDを保存し、エンジニアが作業セッションをまたいで調査を継続できるようにします。あるエンジニアは昨日、レガシー認証モジュールの分析に1時間費やし、そのアーキテクチャと依存関係に関するコンテキストを構築しました。今日も作業を続けたいと考えています。セッションIDは有効ですが、バージョン管理システムを見ると、エージェントが以前に読み取った12個のファイルのうち3個が、チームメイトのマージによって夜間に変更されていました。
効率性と正確性のバランスが最も取れたアプローチはどれか？
A. エージェントが古い前提にとらわれずに最新のコードベースの状態に基づいて動作するように、新しいセッションを開始します。
B. セッションを再開し、対象を再分析するために変更された特定のファイルをエージェントに通知します。
C. セッションを再開し、エージェントに以前に分析した12個のファイルすべてを直ちに再読み込みさせる
D. 変更されたファイルについてエージェントに通知せずにセッションを再開する
Answer: B (メーッセージを残す**)**
再開すると、以前のアーキテクチャ分析、依存関係マッピング、および会話履歴が保持され、1 時間分の有効な作業を繰り返すコストを回避できます。Anthropic は、セッション ID による再開により、読み込まれたファイル、実行された分析、および決定を含むエージェントの以前の完全なコンテキストが復元されると述べています。
(https://code.claude.com/docs)
/en/agent-sdk/sessions)
しかし、セッションはリポジトリのスナップショットではなく、履歴知識を表します。3つのファイルが外部で変更されたため、エージェントはどのファイルが影響を受けたかを伝えられ、以前の結論に依拠する前にそれらを再読み込みするように指示される必要があります。Anthropicは、セッションはファイルシステムではなく会話を永続化することを明確にしています。
https://code.claude.com/docs/en/agent-sdk/sessions) ターゲット再分析は、変更されていない 9 つのファイルの確立されたコンテキストを保持しながら、エージェントのモデルの古い部分を更新します。
オプションAは、分析対象のコードの大部分が変更されていないにもかかわらず、貴重な作業を破棄します。オプションCは正確ですが、変更の有無に関わらず12個すべてのファイルを再処理するため非効率的です。オプションDは安全ではありません。エージェントが、古い関数シグネチャ、依存関係、または制御フローに基づいて推論を継続する可能性があります。
継続プロンプトでは、変更されたファイルを特定し、既知の場合はマージの目的を要約し、以前の理解との比較を重点的に行うよう求めるべきです。これらの変更によって影響を受けるアーキテクチャ上の結論は、明示的に修正する必要があります。
公式の参考資料／トピック :セッション再開、ファイルシステムと会話状態の比較、対象を絞った再分析、リポジトリ変更の認識。

最新問題**: 11**
Claude Agent SDK を使用して、開発者の生産性向上ツールを構築しています。このエージェントは、エンジニアが馴染みのないコードベースを探索したり、レガシーシステムを理解したり、定型コードを生成したり、反復作業を自動化した

りするのに役立ちます。組み込みツールである Read、Write、Bash、Grep、Glob を使用し、Model Context Protocol (MCP) サーバーと統合します。
チームに最近加わったエンジニアが、セキュリティ強化を行う前に、認証と認可のアーキテクチャについてエージェントに説明を求めました。コードベースには、複数のサービスにまたがる800以上のファイルが含まれています。
文脈上の制約を尊重しつつ、理解を深める上で最も効果的な探究戦略とはどのようなものだろうか？
A. 並列サブエージェントを起動してすべてのサービスを同時に調査し、その結果を統合してアーキテクチャの概要を作成します。
B. ファイル名または内容に auth、login、permission、token が含まれるすべてのファイルを読み取ります。
C. まずCLAUDE.mdとREADMEファイルをすべて読み、次にエンジニアに最も重要な認証ファイル10〜15個を特定してもらう。
D. grep を使用して認証エントリポイントを特定し、それらのファイルを読み込み、インポートと関数呼び出しを段階的に追跡して認証フローをマッピングします。
Answer: (解答を表示する**)**
オプションDは、何百ものファイルを無差別に読み込むことなく、証拠に基づいたアーキテクチャマップを作成します。認証エントリポイントには、ルートハンドラ、ミドルウェア登録、トークン検証関数、セッションコンストラクタ、IDプロバイダコールバックなどが含まれる場合があります。Grepがこれらのアンカーを特定した後、ターゲット読み取り操作によってその責任を確立でき、インポート、呼び出しサイト、構成参照によって下流の認証フローが明らかになります。Anthropicの大規模コードベースガイダンスでは、無関係な命令やファイル読み取りがトークンを消費しパフォーマンスを低下させるため、タスクが影響を受けるリポジトリの部分のみにClaudeのスコープを設定することを推奨しています。同社の共通ワークフローガイダンスでも同様に、広範囲から始めて特定のコンポーネントに絞り込むことを推奨しています。オプションAは、関連するサービス境界が不明なうちにエージェントを起動するため、重複した一貫性のない調査が発生する可能性があります。オプションBは、テスト、ドキュメント、無関係なトークン、偶発的な用語を含む、制御されていない語彙的スイープです。オプションCは、プロジェクトガイダンスを適切にチェックしますが、アーキテクチャを知らない可能性のある新しいエンジニアに技術的なファイル選択を誤って委任しています。段階的な依存関係追跡は、コンテキストの効率性を維持しながら、実際のコードパスに基づいた検証可能なエンドツーエンドのマップを生成します。

最新問題**: 12**
自動レビューを導入した後、精度は高いものの再現率が低く、実際のバグが検出されずに見過ごされていることに気づきました。調査の結果、レビュープロンプトでクロードに「確信度の高い問題のみを報告する」コメントしない方が良い」と指示していることが判明しました。開発者はノイズが少ないことを高く評価していますが、レビュー済みのプルリクエストに本番環境の停止を引き起こした競合状態が見られたにもかかわらず、報告されていませんでした。誤検出率を管理可能な範囲に抑えつつ、バグ検出を大幅に改善する必要があります。最も効果的なアプローチは何でしょうか？
A. クロードがフラグを立てるべきバグカテゴリ（競合状態、ヌル参照解除、エラー処理のギャップ）を示す詳細な少数のサンプルを追加し、同時に高信頼性フィルタリング命令を維持します。
B. 保守的な指示を削除し、クロードにすべての潜在的な問題を報告させ、その後、重複する結果を排除し、過去にノイズの多いカテゴリを抑制するプログラムによるフィルターを適用します。
C. レビューを、包括的な網羅を目的とした発見段階（すべての潜在的な問題を信頼度と深刻度のメタデータとともに報告する）と、それらの発見を検証して閾値を設定する別の段階に分割します。

D. 関連するテスト、最近の Git 履歴、モジュールの依存関係グラフを含めるようにコンテキストを拡張し、クロードが深刻度を判断するためのより豊富な証拠を得られるようにします。
Answer: (解答を表示する**)**
プロンプトの保守的な報告ポリシーが、低い再現率の直接的な原因となっている。クロードは分析中に正当な競合状態を発見する可能性があるが、確実な問題のみを報告するという指示を満たせないため、それを抑制してしまう。オプションCは、混同すべきではない2つの目的、すなわち広範な欠陥発見と厳格な受け入れフィルタリングを明確に区別している。Anthropicの現在のコードレビューに関するガイダンスでは、不確実な問題や深刻度の低い問題も含め、すべての問題を報告し、信頼度と深刻度のメタデータを割り当て、別途検証段階でそれらをフィルタリングすることを明示的に推奨しています。これにより、開発者向けのノイズを抑えつつ、問題の再現率を最大化できます。
オプションAでは抑制ポリシーがそのまま維持されるため、追加の例によって発見された問題が報告される保証はありません。オプションBは再現率を向上させますが、過去のカテゴリレベルの抑制に依存しているため、ノイズの多いカテゴリに属する真の発見が破棄される可能性があります。オプションDは分析品質を向上させることができますが、発見を保留する原因となる指示は削除されません。専用の発見ステージに続いて、独立して構成可能な検証または閾値設定ステージを設けることで、競合する目的を最適化するために1つのモデル呼び出しを強制することなく、測定可能な再現率と精度制御を実現できます。

最新問題**: 13**
Claude Agent SDK を使用して、顧客サポート解決エージェントを構築しています。このエージェントは、返品、請求に関する紛争、アカウントの問題など、曖昧さの多いリクエストを処理します。カスタムのモデルコンテキストプロトコル (MCP) ツール (get_customer、lookup_order、process_refund、escalate_to_human) を介して、バックエンドシステムにアクセスできます。目標は、初回接触で 80% 以上の解決率を達成しつつ、エスカレーションが必要なタイミングを判断することです。
運用ログから、エラー処理に一貫性がないことが明らかになりました。lookup_order が失敗した場合、エージェントは 5 回以上再試行することがあり (注文 ID が存在しない場合は無駄です)、すぐにエスカレーションすることがあり (一時的なネットワークの問題の場合は時期尚早です)、また、ユーザーに説明を求めることがあり (問題がバックエンドの権限エラーの場合は不適切です)。調査の結果、MCP ツールは一律のエラー応答を返していることがわかりました: { " isError " : true, " content " : [{ " type " : " text " , " text " : " Operation failed " }]}。エージェントはエラーの種類を区別できません。
最も効果的な改善策は何ですか？
A. エラー応答を構造化メタデータで強化 - error_category (一時的/検証) を含める /permission）、isRetryable ブール値、および失敗の原因の説明。
B. すべてのエラーに対して、MCPサーバーで指数バックオフを使用した再試行ロジックを実装し、再試行が尽きた後にのみエージェントに返します。
C. エラーカテゴリと推奨されるアクションを判断するために、エージェントがエラー発生後に呼び出すanalyze_error MCPツールを作成します。
D. エラーメッセージのパターンを解釈し、それぞれに対して適切な応答を選択する方法を示す、いくつかの例をシステムプロンプトに追加します。
Answer: A (メーッセージを残す**)**

エージェントの動作が一貫していないのは、すべてのエラーが同一の形式で表現されているためです。「操作失敗」というメッセージには、永続性、ユーザーによる修正可能性、認証、または再試行が成功する可能性に関する情報が一切含まれていません。構造化フィールドを追加することで、エラーが実行可能なインターフェース契約に変換されます。
Anthropic は、ツール実装に対して、エラー フラグを使用して失敗をマークし、Claude が修正された入力で再試行したり、明確化を求めたり、制限事項を説明したりするために使用できる情報を返すように指示します。ツールの設計では、モデルが一般的なメッセージから原因を推測することを強制するのではなく、モデルが正しい次のアクションを選択できるように十分な詳細を公開する必要があります。(https://platform.claude.com/docs/en/agents-and-tools/tool-use/build-a-tool-using-agent?
utm_source=chatgpt.com)
error_category、isRetryable、および原因の説明があれば、エージェントは一時的なネットワーク障害を再試行したり、検証エラーの修正済み識別子を要求したり、無意味な繰り返しなしに権限障害をエスカレーションまたは報告したりできます。オプションBは、永続的な検証障害と権限障害を不必要に再試行します。オプションCは、元のツールが既に持っている情報を分類するためだけに、別のツール呼び出しと障害ポイントを追加します。オプションDは、明示的な機械可読な意味論を提供するのではなく、可変テキストと例の解析に依存しています。
このツールは、isError: true を保持し、安定した構造化エラーオブジェクトを返し、適切な場合には安全な顧客向けメッセージを含め、内部の機密情報やスタックトレースを公開しないようにする必要があります。
公式の参照資料／トピック：MCPエラー契約、実行可能なツール結果、再試行可能性メタデータ、エージェントとコンピュータのインターフェース設計。

最新問題: **14**
あなたはClaude Agent SDKを使用して、マルチエージェント型の研究システムを構築しています。コーディネーターエージェントは、専門的なサブエージェントに処理を委任します。サブエージェントは、Web検索、文書分析、調査結果の統合、レポート生成を担当します。このシステムは、様々なトピックを調査し、引用文献付きの包括的なレポートを作成します。
コーディネーターエージェントには、4 つの特殊サブエージェントすべてに対して AgentDefinition オブジェクトが設定されており、それぞれに適切な説明、プロンプト、およびツール制限が設定されています。テスト中に、コーディネーターが委任のタイミングを正しく判断し、「このトピックに関する情報源をウェブ検索エージェントに検索させます」といったメッセージを生成することに気づきますが、サブエージェントの実行は一切行われません。その後、コーディネーターは委任が行われたかのように処理を進め、不完全な情報のまま続行します。ログにはエラーは記録されません。
最も可能性の高い原因は何ですか？
A. AgentDefinition オブジェクトは正しく構成されていますが、コーディネーターのシステムプロンプトに利用可能なサブエージェントの種類が明示的にリストされていないため、モデルはそれらを呼び出すことができることを認識できません。
B. サブエージェントのコンテキスト分離とは、コーディネーターからのタスク記述がサブエージェントに自動的に届かないことを意味します。ClaudeAgentOptions で明示的なコンテキスト転送を設定する必要があります。
C. コーディネーターの allowedTools 設定に Agent (以前は Task と呼ばれていました) が含まれていないため、サブエージェントを生成するために必要なツールを呼び出すことができません。
D. コーディネーターの max_tokens 設定が低すぎるため、サブエージェントの種類を指定する前にサブエージェントツールの呼び出しが切り詰められます。

Answer: C (メーッセージを残す)
オプション C は、委任に関する推論と委任の実行との区別に合致しています。サブエージェントを定義すると、その説明を選択できるようになりますが、コーディネーターは依然として SDK のサブエージェント生成ツールを呼び出す必要があります。現在の Claude Agent SDK のドキュメントでは、これを Agent ツールと呼んでいます。Task は以前の名前で、古い SDK 構成にも関連しています。SDK ドキュメントの Anthropic の Subagents では、サブエージェントの呼び出しが自動的に承認されるように、開発者に AllowedTools に Agent を含めるように指示しています。この権限がない場合、呼び出しは権限コールバックにフォールスルーするか、非対話型権限モードで拒否される可能性があります。オプション A は、構成済みのサブエージェントの説明によって各エージェントをいつ選択すべきかが Claude に既に伝えられているため、可能性は低いですが、明示的なプロンプトによって呼び出しの信頼性を向上させることができます。オプション B はコンテキスト分離を誤って説明しています。コンテキストは生成プロンプトに含める必要がありますが、この問題は呼び出しが試行された後に発生し、すべてのサブエージェントが実行されない理由を説明するものではありません。オプションDでは、ツール呼び出しなしで一貫した口頭での約束が得られるのではなく、通常は切り捨て証拠または不完全な出力が生成されます。したがって、構成ではエージェントを許可し、必要に応じて明示的に委任を要求し、サブエージェントの呼び出しイベントをログに記録する必要があります。

最新問題: **15**
あなたはClaude Codeを継続的インテグレーション／継続的デプロイメント（CI/CD）パイプラインに統合しようとしています。
このシステムは、自動コードレビューを実行し、テストケースを生成し、プルリクエストに対するフィードバックを提供します。実行可能なフィードバックを提供し、誤検出を最小限に抑えるようなプロンプトを設計する必要があります。
パイプラインでは、単一の API 呼び出しを使用してすべてのプルリクエストをレビューします。この呼び出しでは、変更された各ファイルの差分と全文を含む静的なプロンプトが表示されます。変更されていないファイルはレビューに含まれません。開発者からは、レビューでファイル間のバグが見落とされるケースが頻繁に報告されています。たとえば、プルリクエストで関数のパラメータ名が変更されても、古い引数順序を使用している変更されていないファイル内の呼び出し元がレビューで特定されないといったケースです。
評価によると、レビュー済みのプルリクエストに起因する本番環境のインシデントのうち、35%はファイル間バグによるものである。
レビューデザインにおいて、最も効果的な変更点は何でしょうか？
A. 静的依存関係グラフを作成し、変更されたファイルから2つの依存関係ホップ以内にあるすべてのファイルを含めます。
B. モデルに外部参照をリストアップさせ、各変更が未知の呼び出し元にどのように影響するかを段階的に推論するように指示を追加します。
C. レビューを、ファイルの読み込みとリポジトリの検索、参照の追跡によるファイル間の調査結果の検証が可能な、ターン制限のあるエージェントタスクとして再設計します。
D. 変更されたファイルとその直接の依存ファイルごとに個別のレビュー処理を実行し、最終的な統合処理を通じて結果を集約および重複排除します。
Answer: C (メーッセージを残す)
オプションCでは、レビュー担当者はプロンプトに現在表示されていない証拠にアクセスできます。エージェントによるレビューでは、Grep、Glob、Read、言語サーバーツール、およびテストコマンドを使用して、呼び出し元を特定し、イン

ポートを追跡し、型定義を検査し、疑われる互換性の問題が実際に存在するかどうかを確認できます。ターン制限によりコストが抑制されつつ、的を絞った探索が可能になります。
Anthropicのコンテキストエンジニアリングガイダンスでは、ジャストインタイム取得を推奨しています。エージェントは軽量な参照を保持し、大きな固定コンテキストを事前にロードするのではなく、現在のタスクに必要な情報を動的にロードする必要があります。Anthropicはまた、完全なコンテキストを収集するために必要なリポジトリが利用可能になると、コードレビューのパフォーマンスが向上することも報告しています。オプションAでは、リフレクション、生成されたインターフェース、依存性注入、または任意の2ホップ境界を超える間接呼び出しパスが欠落しているにもかかわらず、多くの無関係なファイルがロードされる可能性があります。オプションBでは、モデルが検査できないコードについて確実に推論することはできません。オプションDでは、カバレッジは向上しますが、リポジトリ全体の関係が断片化され、かなりの作業が重複する可能性があります。エージェントループは、差分から開始し、不確実なファイル間の影響を特定し、関連するパスのみを検索し、サポートコードを見つけた後にのみ結果を報告できます。Anthropicコンテキストエンジニアリングガイダンス、Claudeコード品質レポート

最新問題**: 16**
あなたのテスト生成プロセスは新しいコードの単体テストを生成しますが、レビューの結果、55%は価値が低いことが判明しました。
関数が例外をスローしないことだけを検証する些細なアサーション、既存のテストカバレッジを重複させるテスト、あるいはチームのフィクスチャ規約を無視するテストなど。そもそも、価値の低いテストが生成される頻度をどのように減らすべきでしょうか？
A. 過去の品質指標で合格率が高いディレクトリにのみテスト生成を制限し、生成されたテストで常に大幅な編集が必要となる領域ではテスト生成を無効にします。
B. 2段階生成を実装し、2回目のClaude呼び出しで各テストを品質基準に照らして採点し、開発者に提示する前にスコアの低いテストを除外します。
C. CLAUDE.md にテスト基準を文書化します。これには、重要なテスト基準、利用可能なフィクスチャとその使用目的、意味のある動作テストと些細なアサーションを区別する例が含まれます。
D. 生成後のカバレッジ分析を追加し、既存のテストスイートを超えて行カバレッジを増加させない生成されたテストを自動的に除外します。
Answer: C (メーッセージを残す**)**
これらの失敗は、プロジェクト固有の知識が不足していることを反映しています。クロードは、どのフィクスチャが推奨されているか、既存のテストスイートで既にカバーされている内容、チームが意味のある動作とみなしている内容を把握していません。オプションCでは、テスト生成を開始する前に、この情報を永続的なプロジェクトコンテキストとして提供します。これにより、トークンを消費してテストを生成した後に不十分なテストをフィルタリングするのではなく、生成元で生成品質が変更されます。
Anthropic社のCLAUDE.mdドキュメントでは、共有のビルドおよびテストコマンド、コーディング標準、アーキテクチャ上の決定事項、慣例、および一般的なワークフローをプロジェクトのCLAUDE.mdファイルに保存することを推奨しています。テストに関するガイダンスでは、必須の動作アサーション、フィクスチャの選択ルール、重複チェック、命名規則、および許容されるテストと許容されないテストの代表的な例を定義できます。
オプションAは、モデルの理解度を向上させるよりも、複雑なディレクトリを回避することに重点を置き、潜在的に有用な自動化を犠牲にしています。オプションBは、一部の不十分なテストを削除できるかもしれませんが、モデル作業が倍

増し、明示的に記述されるべき品質基準を推測するために、別の確率的呼び出しが必要になります。オプションDは、些細なテストでも意味のある動作を検証することなくカバレッジを向上させることができるにもかかわらず、行カバレッジを品質指標として扱っています。一方、価値のある回帰テストは、既に実行された行をより適切なアサーションでカバーできる可能性があります。したがって、永続的で具体的なテスト標準こそが、最も直接的かつ拡張性の高い改善策となります。

最新問題**: 17**
Claude Agent SDK を使用して、顧客サポート解決エージェントを構築しています。このエージェントは、返品、請求に関する紛争、アカウントの問題など、曖昧さの多いリクエストを処理します。カスタムのモデルコンテキストプロトコル (MCP) ツール (get_customer、lookup_order、process_refund、escalate_to_human) を介して、バックエンドシステムにアクセスできます。目標は、初回接触で 80% 以上の解決率を達成しつつ、エスカレーションが必要なタイミングを判断することです。
運用ログによると、エージェントが6回以上のツール呼び出しを必要とする複雑な請求紛争を処理する場合、データ収集後、解決を完了したりエスカレーションしたりする前に、max_turnsの制限に達してしまうことがある。
チームの目標は、エージェントのループがどのように終了するかに関わらず、すべての顧客とのやり取りが問題解決または担当者への引き継ぎのいずれかで終了することを保証することです。
この保証を実現するには、どの方法が有効でしょうか？
A. ツール呼び出し回数をカウントし、エージェントがmax_turns制限の80%に達したら自動的にエスカレーションしてループを終了する、ツール使用前のフックを実装します。
B. ワークフローを 2 つの連続したエージェント呼び出しに分割します。最初のエージェントは get_customer と lookup_order を介して情報を収集し、次に 2 番目のエージェントがそのデータを受け取り、process_refund または escalate_to_human を処理します。それぞれに個別のターン予算があります。
C. 各ループ終了後にエージェントの結果をチェックするオーケストレーション レイヤー コードを追加します。ループが解決またはエスカレーションを完了せずに終了した場合、蓄積された会話コンテキストとツールの結果を使用してプログラムで escalate_to_human を呼び出します。
D. エージェントが残りのアクションで解決を完了できないと判断した場合、調査結果の概要を添えてescalate_to_humanを呼び出すように指示するシステムプロンプト指示を追加します。
Answer: (解答を表示する**)**
オプションCは、モデルの動作に関わらず、必要な終了条件を保証する唯一のアプローチです。エージェントループが停止すると、オーケストレーションレイヤーは記録された結果を評価します。解決が成功せず、かつ人間によるエスカレーションも確認されなかった場合、決定論的なアプリケーションコードは、終了前に蓄積された情報を使用してescalate_to_humanを呼び出します。

max_turns制限は、ホストが制御する安全境界です。この境界に達すると、モデルはプロンプト指示に従ったり、別のツール呼び出しを行ったりする機会を失う可能性があります。したがって、必要なフォールバックは、残りのターンが尽きる前にClaudeがエスカレーションを決定することに完全に依存することはできません。
オプションAは、任意の80%のしきい値に基づいてエスカレーションを行うため、残りの予算内で解決できたはずの案件を早期に終了させてしまう可能性があります。オプションBは処理能力を向上させますが、2人目の担当者が自身の制限に達する前に作業を完了したり、エスカレーションを行ったりすることを保証するものではありません。オプションDは確率的なものであり、予期せぬ終了が発生した後は動作しません。
オーケストレーションのフォールバックは冪等性を持つべきであり、完了チェックを繰り返しても重複したエスカレーションレコードが作成されないようにする必要があります。また、ハンドオフペイロードには、検証済みの顧客情報、関連する注文データ、完了したアクション、失敗理由、および未解決のリクエストを含める必要があります。
公式参照/トピック: エージェントループの終了、決定論的フォールバックオーケストレーション、端末状態の検証、人間による引き継ぎの保証。

最新問題**: 18**
あなたはClaudeを使用して構造化データ抽出システムを構築しています。このシステムは、非構造化ドキュメントから情報を抽出し、JavaScript Object Notation（JSON）スキーマを使用して出力を検証し、高い精度を維持します。また、エッジケースを適切に処理し、下流システムと統合する必要があります。
モニタリングの結果、抽出処理の12%がPydanticの検証に失敗し、 数量には浮動小数点数が期待されていたが、実際には「2〜3」という値が返された」といった具体的なエラーが発生していることが判明しました。これらのリクエストを修正せずに再試行しても、同様の失敗が発生します。
これらの検証失敗から復旧するための最も効果的なアプローチは何ですか？
A. 検証エラーを含むフォローアップリクエストを送信し、モデルにその出力を修正するように依頼します。
B. 出力のばらつきをなくし、一貫したフォーマットを確保するために、温度を0に設定します。
C. 抽出のために送信する前に、ソース文書を前処理して問題のある形式を標準化します。
D. 検証に失敗したドキュメントを再処理するために、より大きなモデル層を使用したセカンダリパイプラインを実装します。
Answer: A (メーッセージを残す**)**
変更なしの再試行では、同じタスク仕様が繰り返されるため、同じ無効な解釈が再現されることがよくあります。バリデーターは正確な修正情報を生成しました。quantityには浮動小数点数が必要ですが、モデルは範囲文字列「2から3」を返しました。後続の処理でこのエラーを指定すると、一般的な再試行が反復的な修復操作に変換されます。
Anthropic は、反復的な改良を、以前の出力をターゲットを絞った指示とともに後続のリクエストにフィードバックすることで、矛盾を検出して修正する方法として定義しています。（https://docs.anthropic.com/en/docs）/test-and-evaluate/strengthen-guardrails/reduce-hallucinations) オプション A では、そのパターンを直接適用します。Claude は、無効な出力、正確な Pydantic エラー、およびスキーマに準拠した修正を返す指示を受け取ります。アプリケーションは、再試行回数を制限し、元のソースを保持し、情報損失なしに表現できないケースをエスカレーションする必要があります。
オプションBは正しいフォーマットを保証するものではありません。低温では、同じ誤った出力が繰り返し発生する可能性があります。オプションCは、体系的なOCRの問題やソースフォーマットの問題には有効ですが、検証ツールによって

既に直接的なエラーが説明されている場合は、不必要に広範囲に及びます。オプションDは、既存の検証レイヤーから得られる実用的なフィードバックを事前に利用しないまま、コストと複雑さを増大させます。
サポートされているモデルの場合、ネイティブの構造化出力も考慮する必要があります。これは、スキーマに準拠したJSON を保証し、多くの Pydantic の形状エラーが発生する前に防止できるためです。(https://platform.claude.com/docs/en/build-with-claude/structured-outputs)
公式の参考文献/トピック: 反復的改良、検証エラーフィードバック、制限付き再試行ループ、構造化出力。

最新問題: **19**
あなたはClaude Codeを継続的インテグレーション／継続的デプロイメント（CI/CD）パイプラインに統合しようとしています。
このシステムは、自動コードレビューを実行し、テストケースを生成し、プルリクエストに対するフィードバックを提供します。実行可能なフィードバックを提供し、誤検出を最小限に抑えるようなプロンプトを設計する必要があります。
テスト生成によって新しいコードの単体テストが生成されますが、レビューの結果、55%は価値の低いテストであることが判明しました。これは、関数が例外をスローしないことだけを検証する些細なアサーション、既存のカバレッジを重複するテスト、またはチームのフィクスチャ規約を無視したテストなどです。
そもそも、価値の低い検査が生成される割合をどのように減らせばよいのでしょうか？
A. 2段階生成を実装し、2回目のClaude呼び出しで各テストを品質基準に照らしてスコアリングし、開発者に結果を提示する前にスコアの低いテストを除外します。
B. 生成後のカバレッジ分析を追加し、既存のテストよりも行カバレッジが増加しない生成されたテストを自動的に除外します。
C. 過去の品質指標で合格率が高いディレクトリにテスト生成を制限し、生成されたテストで常に大幅な編集が必要な領域ではテスト生成を無効にします。
D. CLAUDE.md にテスト標準を文書化する。これには、重要なテスト基準、利用可能なフィクスチャとその使用例、意味のある動作テストと些細なアサーションを区別する例が含まれる。
Answer: D (メーッセージを残す**)**
オプションDは、後から不良出力をフィルタリングするのではなく、生成条件を改善します。Anthropicは、リポジトリ全体の標準と、Claudeがすべてのセッションで適用すべき情報について、CLAUDE.mdを使用することを推奨しています。このファイルでは、有用なテストの定義、観測可能な動作に関するアサーションの要求、承認済みフィクスチャファクトリの識別、既存シナリオの重複の禁止、強力なテストと単純なテストの対照的な例の提供などが可能です。これらの指示は、Claudeが作成するテストを決定する前に利用できます。
オプション A では、一部の不十分なテストが削除される可能性がありますが、モデル作業が倍増し、最初の段階で低品質な資料が生成される可能性があります。オプション B では、些細なアサーションでも動作を検証せずに新しい行を実行できるにもかかわらず、行カバレッジを品質指標として扱います。オプション C では、生成を改善するのではなく、困難な領域を放棄します。標準では、新しいケースを作成する前に、近くのテストとフィクスチャを検査し、各テストが保護する動作または回帰を特定し、その後、関連するスイートを実行するように Claude に指示する必要があります。CLAUDE.md はセッションごとにコンテキストを消費するため、ガイダンスは簡潔に保つ必要があります。長々としたテスト手順は、プロジェクトの指示から参照される再利用可能なスキルに配置する方が適切です。

最新問題: **20**

あなたはClaude Codeを継続的インテグレーション／継続的デプロイメント（CI/CD）パイプラインに統合しようとしています。
このシステムは、自動コードレビューを実行し、テストケースを生成し、プルリクエストに対するフィードバックを提供します。実行可能なフィードバックを提供し、誤検出を最小限に抑えるようなプロンプトを設計する必要があります。
自動コードレビューでは、プルリクエスト内の実際のバグが見落とされています。調査の結果、レビュープロンプトに「確実に本番環境の障害を引き起こす重大な問題のみをフラグ付けしてください」という指示が含まれていることが判明しました。
些細な懸念事項や不明な点は無視してください。開発者によると、見落とされたバグの中には、モデルが調査したものの報告しなかった真の論理エラーが含まれているとのことです。チームは、レビュー結果が構造化され、各発見事項にメタデータが付与され、実行可能なものであることを求めています。
どの変更を行えば、抑制された結果の原因を取り除き、かつ後続のフィルタリングのために構造化されタグ付けされた出力を維持できるでしょうか？
A. 同じプロンプトを使用して差分を再読み込みし、最初のレビューで見落とされた可能性のあるものを探す、2回目のレビューパスを追加します。
B. モデルに、すべての結果を信頼度と深刻度タグ付きで報告するように指示し、フィルタリングは後続のステップに延期します。
C. プロンプトから重症度に関する指示をすべて削除し、モデルが報告する内容についてデフォルトの判断を使用するようにします。
D. 拡張思考を可能にし、モデルがレビューを生成する前に、すべてのコード変更について段階的に推論するように指示します。
Answer: B (メーッセージを残す**)**
オプションBは、機械可読メタデータを保持しつつ、偽陰性の原因となっているプロンプトレベルの抑制を解除します。Anthropicの現在のコードレビュープロンプトガイダンスでは、「重大な問題のみを報告する」や「慎重に行う」といった指示が文字通りに解釈される可能性があると警告しています。つまり、Claudeは分析中に真の欠陥を特定しても、出力から除外してしまう可能性があるということです。Anthropicは、すべての検出結果を要求し、フィルタリングを個別に適用することを推奨しています。
信頼度と深刻度フィールドを使用すると、下流のコードで調整可能なしきい値を適用できますが、モデルが生成中に証拠を破棄することを強制することはできません。スキーマでは、ファイル、行、説明、深刻度、信頼度、証拠、推奨アクションなどのフィールドを要求できます。Anthropic の構造化出力ドキュメントは、このような応答契約の強制をサポートしています。オプション A は同じ抑制指示を繰り返し、同じ省略を再現する可能性が高いです。オプション C は明示的なレポート構造を削除し、フィルタリング動作を未定義のままにします。オプション D は分析の深さを向上させる可能性がありますが、拡張推論は不確実な結果を抑制する直接指示を上書きしません。検出と決定論的フィルタリングを分離することで、再現率、構造、および運用制御が維持されます。

最新問題**: 21**
あなたはソフトウェア開発を加速するためにClaude Codeを使用しています。あなたのチームは、コード生成、リファクタリング、デバッグ、ドキュメント作成にClaude Codeを利用しています。カスタムスラッシュコマンドやCLAUDE.mdの設定を用いて開発ワークフローにClaude Codeを統合し、プランモードと直接実行の使い分けを理解する必要があります。

あなたのチームはClaude CodeでMCPサーバーの設定を行っています。チームメンバー全員がアクセスできる共有会場検索サーバーを追加したいと考えており、さらにあなた自身がテストする実験的な音楽プレイリストサーバーも追加したいと考えています。
どの構成方法がMCPサーバーのスコープを正しく適用しますか？
A. 両方のサーバーをローカルの ~/.claude.json に追加します。
B. 会場サーバーを .mcp.json に、プレイリストサーバーを ~/.claude.json に追加します。
C. ~/.claude.json に会場サーバーを、.mcp.json にプレイリストサーバーを追加します。
D. 両方のサーバーをプロジェクトレベルの.mcp.jsonファイルに追加します。
Answer: (解答を表示する**)**
会場サーバーはすべてのプロジェクト貢献者が利用できる必要があるため、プロジェクトスコープに含まれます。Claude Codeは、プロジェクトスコープのMCP設定をリポジトリのルートにある.mcp.jsonファイルに保存します。このファイルはバージョン管理システムにコミットしてチームと共有できます。
実験用プレイリストサーバーは非公開のままにしておく必要があります。ローカルスコープの MCP サーバーは ~/.claude.json の現在のプロジェクトのエントリの下に保存されますが、ユーザースコープのサーバーも ~/.claude.json に保存されますが、すべてのプロジェクトでロードされます。このサーバーは個人的な実験として説明されているため、どちらのプライベートスコープも概念的には適切です。リストされている選択肢の中で、オプション B は共有プロジェクト構成とプライベートな個人構成を正しく分離しています。(https://code.claude.com/docs/en/mcp) オプション A はチームが会場構成を受け取れないようにします。オプション C は意図した可視性を逆にします。
オプションDでは、実験的なサーバー構成が他の貢献者に公開され、マシン固有のコマンドや認証情報がコミットされる可能性があります。
推奨されるコマンドは、共有会場サーバーの場合は --scope project を、現在のリポジトリに限定した実験の場合は --scope local を指定します。プロジェクトスコープのサーバーでは、Claude Code が .mcp.json で受信した構成を有効にする前に、ユーザーの承認も必要です。
公式の参照/トピック: MCP インストール スコープ; プロジェクト スコープ; ローカル スコープ; .mcp.json ; ~/.claude.json 。

最新問題**: 22**
Claude Agent SDK を使用して、顧客サポート解決エージェントを構築しています。このエージェントは、返品、請求に関する紛争、アカウントの問題など、曖昧さの多いリクエストを処理します。カスタムのモデルコンテキストプロトコル (MCP) ツール (get_customer、lookup_order、process_refund、escalate_to_human) を介して、バックエンドシステムにアクセスできます。目標は、初回接触で 80% 以上の解決率を達成しつつ、エスカレーションが必要なタイミングを判断することです。
担当エージェントが請求に関する紛争を処理しています。get_customerとlookup_orderを呼び出した後、紛争はプロモーション価格の誤りに関連しており、エージェントの承認レベルを超えるマネージャーの承認が必要であることが判明しました。
ワークフローは、このプロセス途中のエスカレーションにどのように対応すべきでしょうか？
A. escalate_to_human を呼び出し、顧客の元のメッセージのみを渡します。
B. escalate_to_human を呼び出す前に、顧客の詳細、注文情報、および特定された問題を含む構造化された引き継ぎを作成します。

C. システムがトランザクションを拒否した場合にのみエスカレーションし、process_refund を使用してとにかく返金を試みます。
D. 会話とツール応答の履歴全体をデータベースに保存し、参照IDを使用してescalate_to_humanを呼び出します。
Answer: B (メーッセージを残す**)**
プロセス途中でエスカレーションが発生した場合は、エージェントが蓄積した意思決定準備完了状態を人間が引き継ぐ必要があります。人間の審査担当者は、確認済みの顧客ID、関連する注文情報、プロモーション価格の不一致、承認が必要な理由、および既に試みられたアクションを把握する必要があります。オプションBでは、この情報を簡潔かつ構造化された形で引き継ぎ、完全な生のトランスクリプトを不必要に繰り返すことを回避します。
Anthropic のツール設計ガイドラインでは、Claude または次のワークフロー参加者が次のアクションを決定するために必要な情報のみを含む、高シグナル情報と安定した識別子を返すことを推奨しています。Anthropic のコンテキストエンジニアリングガイドラインも同様に、すべての冗長なツール結果を保持することなく、重要な状態と依存関係を保持する構造化されたメモを推奨しています。構造化されたエスカレーションペイロードは、これらの両方の原則を適用し、マネージャーの処理時間を短縮します。（https://platform.claude.com/docs/en/agents-and-tools）
/tool-use/define-tools ）
オプション A は既に完了した調査を破棄します。オプション C はエージェントの権限境界を侵害し、許可されていない金融行為のリスクがあります。オプション D は監査可能性を提供しますが、参照 ID だけでは、人間が過度に広範な記録からケースを再構築する必要があります。エージェントが権限外の決定に遭遇した場合、人間の制御は意味のあるものでなければなりません。エージェントは一時停止し、十分な裏付けとなるコンテキストとともに決定を返還する必要があります。（https://www.anthropic.com/research/trustworthy-agents ）公式参照トピック: 構造化されたエージェントの引き継ぎ、高シグナルツールの結果、人間の制御境界、永続的な構造化状態。

最新問題**: 23**
あなたはClaudeを使用して構造化データ抽出システムを構築しています。このシステムは、非構造化ドキュメントから情報を抽出し、JSONスキーマを使用して出力を検証し、高い精度を維持します。また、例外的なケースにも適切に対応し、下流システムと統合する必要があります。
貴社の抽出システムは、2種類の文書を処理します。1つは処理後にアーカイブされる標準的な月次報告書、もう1つは受信後30分以内に業務アラートを発動する必要がある緊急例外報告書です。
どちらも同じJSONスキーマを使用します。レイテンシ要件を満たしつつ、APIコストを最小限に抑えたいと考えています。
処理パイプラインはどのように設計すべきでしょうか？
A. 追跡のために、すべてのドキュメントをカスタムID値とともにメッセージバッチAPIに送信します。結果が到着したら、緊急ドキュメントを即座に処理し、例外が発生した場合は遅延アラートをトリガーします。
B. 標準レポートはメッセージバッチAPIにルーティングしてコストを50%削減し、緊急例外レポートはリアルタイムメッセージAPIにルーティングします。
C. すべてのドキュメントをキューに入れ、1時間ごとにバッチを送信し、バッチの結果が返されたときに緊急ドキュメントにフラグを付けて迅速に処理します。
D. すべてのドキュメントをリアルタイムメッセージAPIに送信して、ドキュメントの種類に関わらず一貫した処理遅延を確保します。
Answer: B (メーッセージを残す**)**

オプションBは、非同期の一括処理とレイテンシに敏感な運用処理を適切に分離しています。Anthropicは、Message Batches APIを、即時応答を必要としない大量リクエスト向けの費用対効果の高い非同期サービスと説明しています。バッチリクエストは個別に処理され、通常は1時間以内に完了し、最大24時間保留状態になる場合があり、標準API価格の50%で課金されます。これらの特性は、処理後にアーカイブされ、差し迫った業務上の期限がない標準的な月次レポートに適しています。ただし、緊急の例外レポートは30分以内にアラートをトリガーする必要があります。バッチサービスは、多くのバッチがすぐに完了したとしても、この期限を保証することはできません。ワークロードは、通常の完了時間をサービスレベルの保証として使用すべきではありません。例外を同期Messages API経由でルーティングすることで、アラートに必要なリアルタイム応答パスが提供され、共有JSONスキーマによって、両方のルートで下流のレコードの一貫性が維持されます。オプションAは、アラートの遅延を意図的に許可しています。オプションCは、処理が開始される前に1時間ごとの送信遅延を導入します。オプションDはレイテンシ要件を満たしていますが、緊急性のないレポートでバッチ割引を無駄にしています。したがって、文書の優先順位に基づくルーティング層は、抽出契約を変更することなく、コスト目標とビジネス期限の両方を満たす。

最新問題**: 24**
あなたは、プルリクエストを分析して構造化された結果を返す、LLM を利用したコードレビューツールを開発しました。各結果は、ファイルパス、行番号、問題カテゴリ (セキュリティやスタイルなど)、および説明を含む JSON オブジェクトです。開発者は役に立たないと判断した結果を却下できますが、現在、35% の結果が却下されています。あなたは、これらの却下を分析して、システムが何を誤っているのかを理解し、それに応じてプロンプトを改善したいと考えています。この分析を最も効果的にサポートするには、出力構造をどのように変更すればよいでしょうか？
A. model_confidence フィールドを 0.0 から 1.0 まで追加し、過去の解雇率に基づいて調整された閾値以下の結果をフィルタリングします。
B. 検出をトリガーした特定のコード構造（例えば、1文字のループ変数など）を記録する detected_pattern フィールドを追加します。
C. 説明欄を拡張して、各問題がなぜ重要なのか、そしてどのように解決すべきかについて、より詳細な説明を追加してください。
D. issue_category フィールドを削除し、個人発見レベルでのみ却下率を追跡します。
Answer: B (メーッセージを残す**)**
オプションBは、繰り返し発生する誤検出パターンを理解するために必要な、より詳細なエラー分類体系を作成します。既存のissue_categoryフィールドは範囲が広すぎるため、スタイルに関する高い却下率だけでは、開発者が1文字の変数、行長の警告、命名規則、あるいはその他の何かに異議を唱えているのかどうかが分かりません。検出された構造を記録することで、却下率をトリガーごとにグループ化し、対象となるプロンプトの修正やプロジェクト固有の例に関連付けることができます。
Anthropicの評価ガイドラインでは、実際の運用状況や特殊なケースを反映した、測定可能でタスク固有の基準と評価事例を推奨しています。また、構造化された出力ドキュメントは、この種のダウンストリーム分析に適した、スキーマ制約付きで解析可能なレコードをサポートします。
オプションAは発見事項の順位付けに役立ちますが、信頼度スコアでは開発者がそれらを却下する理由が説明されず、問題の種類によって調整が不十分になる可能性があります。オプションCは集計のための安定した次元を追加せずにテキスト量を増やします。オプションDは有用なカテゴリ情報を削除し、体系的な分析を困難にします。正規化された

detected_patternフィールドを使用すると、ダッシュボード、パターンレベルの却下指標、代表例のサンプリング、およびプロンプト変更の制御された評価が可能になり、より上位のレポート用に広範なカテゴリが保持されます。

最新問題: **25**
Claude Agent SDK を使用して、開発者向け生産性向上ツールを構築しています。このエージェントは、エンジニアが馴染みのないコードベースを探索したり、レガシーシステムを理解したり、定型コードを生成したり、反復作業を自動化したりするのに役立ちます。組み込みツール（読み取り、書き込み、Bash、Grep、Glob）を使用し、Model Context Protocol（MCP）サーバーと統合します。
1.5 エンジニアは、新しいキャッシュ無効化トリガーを追加する前に、エージェントにキャッシュレイヤーの仕組みを理解するよう依頼します。エージェントは、最初の grep 検索の後、キャッシュロジックがデコレータ、ミドルウェア、サービス クラスを含む 15 個のファイル (合計約 6,000 行) にまたがっていることを特定しました。
状況の制約を管理しながら理解を深めるための、最も効果的な次のステップは何でしょうか？
A. grep を使用して、すべてのファイルから「invalidate」と「expire」のパターンを検索し、最小限の周辺コンテキストで特定の行範囲のみを読み取ります。
B. Readツールを使用して15個のファイルをすべて順番に読み込み、キャッシュの実装全体について完全に理解を深めます。
C. Glob を使用して、一般的なキャッシュパターン (cache*.py、caching/) に一致するファイルを見つけ、最初に最大のファイルを読み込むことで優先順位を付け、次に小さなファイルのギャップを確認します。
D. インポートとクラス階層を分析して、基本キャッシュクラスを特定します。そのファイルを読み込んでインターフェースを理解し、具体的な無効化の実装を追跡します。
Answer: D (メーッセージを残す)
正しい目標は、完全な実装を利用する前に、アーキテクチャマップを作成することである。
基本キャッシュ抽象化、そのインターフェース、およびそれを実装または呼び出すクラスを特定することで、エージェントはコード内を依存関係に基づいて追跡できるようになります。これにより、提案されたトリガーに関連する無効化の実装と統合ポイントのみを検査することが可能になります。
このアプローチはコンテキストウィンドウを保護します。Anthropicは、読み込まれるすべてのファイルがコンテキストを占有し、ウィンドウがいっぱいになるとモデルのパフォーマンスが低下する可能性があると述べています。同社のClaude Codeガイダンスは、多数のファイルを読み込む無制限の調査に対して警告を発し、調査範囲を狭めるか、または調査を委任することを推奨しています。
（https://code.claude.com/docs/en/best-practices）
オプションAは語彙検索に偏りすぎている。invalidateやexpireのみで検索すると、イベント駆動型の無効化、オーバーライドされたメソッド、キャッシュキーの変更、汎用インターフェース呼び出しを見落とす可能性がある。オプションBは、関連性を事前に判断することなく約6,000行を読み込む。オプションCは、ファイル名のパターンとファイルサイズがアーキテクチャ上の重要性と相関していると想定している。最大のファイルには付随的なコードが含まれている可能性があり、小さなインターフェースが全体の設計を定義している場合もある。
オプションDは、任意のファイル順序ではなく、制御関係と型関係に従います。エージェントは基底クラスを読み込んだ後、サブクラス、インポート、構築箇所、ミドルウェアフック、無効化契約への呼び出しを検索し、証拠が必要な場合にのみ段階的に拡張します。

公式の参考文献／トピック :コンテキスト効率の良い探索、依存関係に基づいた読解、アーキテクチャインターフェース、範囲を絞った調査。

最新問題: **26**
自動レビューツールは、セキュリティ問題、API設計、ビジネスロジックの正しさを網羅する単一のプロンプトを使用します。評価スイートでは、API設計に関する検出結果の再現率は82%と高いものの、クイズ採点におけるビジネスロジックのエッジケースの再現率は34%と低いことが示されています。プロンプトにロジックバグのサンプルをいくつか追加すると、ロジックの再現率は41%に向上しますが、API設計の再現率は68%に低下します。このトレードオフに対処して、両方のカテゴリにおける検出率を向上させるにはどうすればよいでしょうか？
A. 変更されたファイルと周辺コードだけでなく、リポジトリ全体をコンテキストとして提供することで、モデルがビジネスロジックパターンをより深く理解できるようになります。
B. 少数の例を、スコア計算におけるゼロ除算や採点閾値における境界条件など、検証すべき具体的な論理エッジケースの詳細なチェックリストに置き換えます。
C. レビューをセキュリティとAPI設計に関するものとビジネスロジックに関するものの2つの焦点を絞った別々のプロンプトに分割し、それぞれに専用の例を用意し、投稿する前に調査結果を統合します。
D. より高性能なモデルティアにアップグレードしてください。より強力な推論により、単一のプロンプトで両方の懸念タイプを処理し、リコールのトレードオフを解消します。
Answer: (解答を表示する**)**
評価結果は、即時的な干渉を示しています。ビジネスロジックの例への注意を高めると、API設計のパフォーマンスが低下します。オプションCは、競合する目的を分離することで、各モデル呼び出しに焦点を絞った用語、例、証拠要件、および評価基準を適用できるようにします。結果として得られる構造化された調査結果は、公開前に統合、重複排除、およびランク付けされます。
Anthropic社の 効果的なAIエージェントの構築」では、タスクの独立した側面を個別のモデル呼び出しで処理し、その後集約するセクショニングによる並列化について説明しています。このパターンは、1つのプロンプトで複数の異なる考慮事項を評価する必要があり、単一の呼び出しではそれらすべてを確実に処理できない場合に適しています。
オプションAは、レビュー目的間の競合を解消することなくコンテキスト量を増加させます。無関係なリポジトリコンテンツは、さらに注意力を分散させる可能性があります。オプションBは、既知のクイズ採点ケースを改善する可能性がありますが、レビュー担当者を不完全なチェックリストに過剰適合させる可能性があり、API設計の想起を回復しません。
オプションDは、追加のモデル機能によってプロンプトの干渉が排除されると想定していますが、評価の裏付けとなる証拠はありません。
個別の専門的なレビュープロセスは、測定された不具合に直接対処する。各プロセスは個別に評価され、その後、統合によって再現性が維持され、重複した所見が生じないことを確認するエンドツーエンドの評価が行われるべきである。

最新問題: **27**
あなたはClaude Codeを継続的インテグレーション／継続的デプロイメント（CI/CD）パイプラインに統合しようとしています。
このシステムは、自動コードレビューを実行し、テストケースを生成し、プルリクエストに対するフィードバックを提供します。実行可能なフィードバックを提供し、誤検出を最小限に抑えるようなプロンプトを設計する必要があります。

自動レビューパイプラインの初期テスト中に、変更されたファイルが50個以上含まれる大規模なプルリクエストのレビューに20分以上かかり、実行ごとに8〜12ドルの費用がかかる場合があることに気づきました。これは、エージェントによるループ処理が頻繁に発生するためです。Claudeはファイルを読み込み、分析ツールを実行し、何度も繰り返します。チームは、各呼び出しが一定の反復回数と一定の金額に達した時点で、周囲のジョブランナーではなくClaude Code自体によって強制的に中断されるようにする必要があります。
どの設定変更が、呼び出しごとの上限値を直接的に適用しますか？
A. --model フラグをより小さく、より安価なモデルに変更することで、各イテレーションで使用するトークンの数を減らし、呼び出しごとのコストを低くします。
B. GitHub Actionsジョブステップでtimeout-minutes: 5を設定し、Anthropic Consoleの使用状況ダッシュボードを通じて実行ごとのコストを監視します。
C. 反復回数と支出を制限するために、claude -p の呼び出しに --max-turns 10 --max-budget-usd 2.00 を追加します。
D. --permission-mode dontAsk を設定すると、明示的に許可されたセットに含まれていないツール権限要求が自動的に拒否されます。
Answer: (解答を表示する**)**
オプションCは、質問で要求されている2つのネイティブ停止制御を適用します。AnthropicのClaude Code CLIリファレンスでは、--max-turnsを印刷モードで許可されるエージェントの最大ターン数、--max-budget-usdをその呼び出しにおけるAPIの最大支出額と定義しています。ターン制限に達すると、実行はエラーで終了します。サブエージェントによる支出は予算上限にカウントされ、現在のClaude Codeバージョンでは、上限に達すると残りのバックグラウンドサブエージェントが停止します。したがって、これらのフラグは、単一のモデル要求だけでなく、呼び出し全体を制御します。
オプションAは、反復ごとの予想コストを削減しますが、反復回数または総支出額に上限を設けません。オプションBは、CIランナーレベルでの実時間実行を制限し、支出を事後的にのみ監視します。クロードコードの予算は適用されません。オプションDは、反復回数、実行時間、トークンの支出ではなく、権限の動作を制御します。パイプラインは両方のフラグを使用し、結果として生じるゼロ以外の終了ステータスまたは構造化エラーステータスをキャプチャし、レビューが正常に終了したか、設定された運用制限によって終了したかを報告する必要があります。

最新問題**: 28**
あなたはClaude Codeを継続的インテグレーション／継続的デプロイメント ⑭I/CD）パイプラインに統合しようとしています。
このシステムは、自動コードレビューを実行し、テストケースを生成し、プルリクエストに対するフィードバックを提供します。実行可能なフィードバックを提供し、誤検出を最小限に抑えるようなプロンプトを設計する必要があります。
パイプラインは以下を実行します。
PROMPT= 'あなたはコードレビュー担当者です。提供された差分を分析して、バグ、セキュリティ上の問題、およびスタイル違反がないか確認してください。
'
クロード -p \
--危険な権限スキップ \
--system-prompt " $PROMPT " \
< diff.txt

レビューは完了してフィードバックを返しますが、Claude はパイプで渡された差分テキストにしかコメントせず、差分が他の多くのモジュールから呼び出される関数を変更する場合でも、チェックアウトされたリポジトリ内の周辺ファイルを読み込んでより広いコンテキストを理解することはありません。
呼び出し方法をどのように変更すれば、カスタムレビュー手順を適用しつつ、Claudeが関連するリポジトリファイルを検査するようになりますか？
A. --system-prompt は -p でのツール使用と互換性がないため、--system-prompt を完全に削除し、レビュー手順を CLAUDE.md ファイルに記述します。
B. --system-prompt を維持し、--allowedTools " Read,Glob,Grep " を追加します。そうしないと、非対話型の -p モードではファイルシステム ツールが無効になります。
C. 差分を標準入力にパイプするのをやめてプロンプト内に埋め込むことで、Claude Code が呼び出しをエージェント セッションとして扱うようにします。
D. --system-prompt を --append-system-prompt に置き換え、より広範なコンテキストが必要な場合に、関連するリポジトリ ファイルを検査するように Claude に明示的に指示します。
Answer: (解答を表示する**)**
オプション D は、Claude Code の標準的なコーディング エージェントの指示を維持しつつ、特殊なレビュー基準を追加します。Anthropic のドキュメントによると、--system-prompt は、ツール ガイダンス、安全に関する指示、コーディング規約など、デフォルトのシステム プロンプト全体を置き換えます。技術的にはツールを無効にするわけではありませんが、ガイダンスを削除すると、呼び出しが狭い範囲のテキスト プロセッサのように動作する可能性があります。--append-system-prompt は、デフォルトの動作を維持し、その上にレビュー指示を重ねます。
プロンプトは、diffだけでは不十分な場合に、ClaudeにRead、Glob、Grepを使用して定義、呼び出し元、テスト、および関連モジュールを検査するように明示的に指示する必要があります。オプションAは、--system-promptがツールと互換性がないわけではないため、誤りです。オプションBも不正確です。--allowedToolsはツールの実行を事前に承認するものであり、-pによってツールが無効になる場合にツールを使用可能にするものではありません。このコマンドでは、--dangerously-skip-permissionsによって既に権限プロンプトがバイパスされています。オプションCは、Claude Codeが非対話モードでパイプされた標準入力を公式にサポートしているため、誤りです。したがって、修正された呼び出しでは、--append-system-promptを使用し、リポジトリ探索の要件を明示的に含める必要があります。Claude Codeのプログラムによる使用法、CLIシステムプロンプトのリファレンスを参照してください。

最新問題**: 29**
Claude Agent SDK を使用して、開発者の生産性向上ツールを構築しています。このエージェントは、エンジニアが馴染みのないコードベースを探索したり、レガシーシステムを理解したり、定型コードを生成したり、反復作業を自動化したりするのに役立ちます。組み込みツールである Read、Write、Bash、Grep、Glob を使用し、Model Context Protocol (MCP) サーバーと統合します。
エンジニアは、認証がサービス全体でどのように使用されているかを理解するために、@company/auth パッケージをインポートするモノレポ内のすべてのファイルをエージェントに検索するように依頼します。
この作業に最も適した組み込みツールはどれですか？
A. package.json ファイルから始めて、依存関係の宣言を追跡します。
B. Glob、ファイル名またはパスにauthを含むファイルを検索します。
C. grepを使用して、ファイルの内容をimport文のパターンで検索します。

D. Bash で find . -type d -name " *auth* " を実行して、一致するディレクトリを探索します。
Answer: C (メーッセージを残す**)**
オプション C は、ソース ファイルの内容を直接検索して @company/auth のインポートを探します。Anthropic の Claude Code ツール リファレンスによると、Grep はファイル内のパターンを検索し、Glob はファイル名とパスを照合します。検索では、静的インポート、require() 呼び出し、動的インポート、再エクスポートなど、リポジトリでサポートされているモジュール構文を考慮できます。また、必要に応じて、ファイル タイプまたはディレクトリで結果を制限することもできます。オプション A は、依存関係を宣言しているパッケージを特定しますが、どのソース ファイルがそれをインポートまたは呼び出しているかは証明しません。ワークスペース レベルの依存関係宣言は、多くのパッケージをカバーしている可能性があります。オプション B は、認証関連のファイル名を検索しますが、これはファイルがパッケージをインポートしているかどうかとは関係ありません。オプション D はディレクトリ名を検索するため、同じ制限があります。Grep が一致するファイルと行番号を返した後、エージェントは周囲のコードを読み取って、各インポートがミドルウェア登録、トークン検証、認可チェック、または再エクスポートをサポートしているかどうかを判断できます。これにより、コンテキストの消費と不要なファイルの読み取りを最小限に抑えながら、正確な使用状況インベントリが生成されます。

最新問題**: 30**
あなたはClaude Agent SDKを使用して、マルチエージェント型の研究システムを構築しています。コーディネーターエージェントは、専門的なサブエージェントに処理を委任します。サブエージェントは、Web検索、文書分析、調査結果の統合、レポート生成を担当します。このシステムは、様々なトピックを調査し、引用文献付きの包括的なレポートを作成します。
ウェブ検索と文書分析のサブエージェントがタスクを完了したら、コーディネーターは分析結果を統合するための合成サブエージェントを起動する必要があります。
合成サブエージェントに必要な情報を提供するための正しいアプローチは何ですか？
A. サブエージェントに、コールバックを通じて他のサブエージェントから出力を要求できるようにするツール定義を提供します。
B. 両方のサブエージェントからの完全な結果を合成サブエージェントのプロンプトに直接含めます。
C. コーディネーターからの自動コンテキスト継承に依存し、簡単なタスク説明のみでサブエージェントをスポーンします。
D. 参照識別子を渡し、他のサブエージェントが結果を格納した共有メモリ ストアへの読み取りアクセス権を持つようにサブエージェントを設定します。
Answer: B (メーッセージを残す**)**
オプション B は、Claude Agent SDK のサブエージェント コンテキスト モデルに従います。通常のサブエージェントは新しいコンテキスト ウィンドウから開始し、親エージェントの会話履歴や以前のツール結果を継承しません。SDK ドキュメントの Anthropic のサブエージェントに関する説明では、親からサブエージェントに渡される情報は、起動ツールのプロンプト ストリングであると述べられています。したがって、必要なファイル パス、決定、エラー、または発見事項は、そのプロンプトに含める必要があります。このシナリオでは、コーディネーターは、両方のエージェントの関連する発見事項、ソース メタデータ、および明示的な合成指示を提供する必要があります。 完全な発見事項」とは、すべての中間検索トレースではなく、必要な結果アーティファクト全体を意味します。オプション C は、コンテキストの自動継承を誤って想定しています。オプション A は、必要でもなく、標準的なハンドオフ メカニズムでもないコールバックを導入し

ています。オプション D は、共有ストアが意図的に実装されている場合は有効なカスタム アーキテクチャになり得ますが、質問ではそのようなインフラストラクチャは確立されておらず、識別子だけではサブエージェントに情報を提供することはできません。プロンプトでは、ウェブ上の調査結果、文書上の調査結果、情報源、未解決の矛盾点、および期待される出力を区別するために、明確なセクションまたは構造化されたオブジェクトを使用する必要があります。これにより、文脈の分離が維持されつつ、統合タスクが実際に必要とするすべての情報が提供されます。

最新問題: **31**
自動レビューCIジョブの初期化に18秒かかり、その後Claudeがコード分析を開始します。プロファイリングの結果、この遅延は、モノレポ全体に存在するフック、MCPサーバー、プラグイン、スキル、および複数のネストされたCLAUDE.mdファイルを自動的に検出することに起因していることが判明しました。起動時間を短縮しつつ、ルートレベルのCLAUDE.mdファイルに記述されているチームのコーディング標準をレビューが確実に遵守するようにする必要があります。
最も効果的なアプローチは何でしょうか？
A. --bare モードで実行し、外部ファイルを参照せずに、すべての CI 呼び出しに対して -p プロンプト引数ですべてのレビュー基準を直接指定します。
B. --system-prompt-file ./CLAUDE.md を使用してデフォルトのプロンプトを完全に置き換えます。これにより、デフォルトのプロンプトアセンブリがバイパスされ、プロジェクトルールのみが読み込まれます。
C. --bare モードで実行し、--append-system-prompt-file ./CLAUDE.md を渡すと、自動検出をすべてスキップしてプロジェクト標準を明示的にロードできます。
D. デフォルトの初期化を維持し、--exclude-dynamic-system-prompt-sections を追加して、マシンごとのプロンプトのばらつきを減らし、ランナー全体でプロンプト キャッシュのヒット率を向上させます。
Answer: C (メーッセージを残す**)**
Claude Code の --bare オプションは、スクリプトの実行速度を向上させるために特別に設計されています。このオプションを使用すると、CLAUDE.md ファイル、フック、スキル、プラグイン、MCP サーバー、および自動メモリの自動検出がスキップされますが、Bash、ファイル読み取り、ファイル編集などの基本的な組み込み機能は維持されます。--bare オプションはルート CLAUDE.md の自動読み込みも防止するため、必要な標準規格は明示的に追加する必要があります。オプション C は両方の目的を達成します。--append-system-prompt-file ./CLAUDE.md は、Claude Code のデフォルトのコーディング エージェント システム プロンプトとツール使用ガイダンスを維持しながら、プロジェクト標準を現在の呼び出しにロードします。公式の Claude Code CLI リファレンスによると、append フラグはファイルの内容をデフォルトのシステム プロンプトに追加しますが、replacement フラグはそのデフォルトのガイダンスを破棄します。
オプションAは機能的には動作するものの、すべてのコマンドで標準規格が重複し、設定のずれが生じます。オプションBはデフォルトのプロンプト全体を置き換え、有用なコーディング、安全対策、ツール使用に関する指示を削除します。オプションDは異なるマシン間でのプロンプトキャッシュの再利用性を向上させますが、フック、プラグイン、スキル、MCPサーバー、CLAUDE.mdの検出を無効にするわけではないため、測定された初期化遅延に直接対処するものではありません。

最新問題: 32
あなたはClaudeを使用して構造化データ抽出システムを構築しています。このシステムは、非構造化ドキュメントから情報を抽出し、JavaScript Object Notation（JSON）スキーマを使用して出力を検証し、高い精度を維持します。また、エッジケースを適切に処理し、下流システムと統合する必要があります。
パイプラインでは、論文の詳細情報を格納するJSONスキーマを持つextract_metadataというツールを使用しています。また、メタデータの充実化のためにlookup_citationsとverify_doiというツールも定義しています。テスト中に、ユーザーが「メタデータを抽出して、引用数を教えてください」といったリクエストを含めると、Claudeが最初にlookup_citationsを呼び出すことがあることに気づきました。しかし、lookup_citationsはextract_metadataが提供するDOIを必要とするため、失敗します。
構造化メタデータの抽出が最初に行われるようにするための最も効果的な方法は何ですか？
A. tool_choice を {"type": "tool", "name": "extract_metadata"} に設定し、抽出されたメタデータを受け取った後、後続のターンでエンリッチメント要求を処理します。
B. tool_choice を "auto" に設定し、ツール定義の順序を変更して extract_metadata がツール配列の先頭に表示されるようにします。これは、Claude がリストの上位にあるツールを優先するためです。
C. パイプライン内のすべての API 呼び出しに対して tool_choice を {"type": "tool", "name": "extract_metadata"} に設定し、エンリッチメントが行われる前に必ず Claude がメタデータを抽出するようにします。
D. tool_choice を "any" に設定して、クロードがツールを使用するようにし、extract_metadata を優先するシステムプロンプトの指示と組み合わせます。
Answer: (解答を表示する)
依存関係は、確率的なツール選択に任せるのではなく、オーケストレーションによって強制する必要があります。Anthropic のドキュメントによると、tool_choice: {"type": "tool", "name": "..."} は、Claude に指定されたツールを呼び出すように強制します。対照的に、auto は Claude が呼び出すツールとその種類を決定できるようにしますが、any は何らかのツールを必要としますが、特定のツールを強制するわけではありません。
（https://platform.claude.com/docs/en/agents-and-tools/tool-use/define-tools）したがって、オプション A は決定論的な 2 段階のワークフローを確立します。最初の API ターンで extract_metadata が強制されます。
DOIやその他の構造化された論文詳細を生成する。アプリケーションはその結果を検証して保存する。次のターンでは、verify_doiとlookup_citationsを公開または許可し、抽出されたDOIを明示的な状態として渡す。この設計により、暗黙的なツール依存性がアプリケーション制御の実行グラフに変換される。
オプションBは、配列の順序が文書化された優先順位メカニズムではないため、選択を保証することはできません。オプションCは、エンリッチメントが行われるべきターンを含め、すべての呼び出しでextract_metadataを強制的に実行するため、無限ループまたは進行しないワークフローが発生する可能性があります。オプションDは、利用可能なツールが1つだけ呼び出されることを保証するだけであり、DOIが存在する前にClaudeがlookup_citationsを選択する可能性があります。
入力データの整合性をより高めるため、ツールは厳密なスキーマを使用し、引数が宣言されたJSONスキーマに準拠するようにすることもできます。ただし、シーケンス要件はオーケストレーション層の責任となります。

公式の参考資料／トピック :ツールの選択、強制的なツール呼び出し、複数ターンにわたるツールオーケストレーション、ツールの依存関係管理。

最新問題: **33**
あなたはClaudeを使用して構造化データ抽出システムを構築しています。このシステムは、非構造化ドキュメントから情報を抽出し、JavaScript Object Notation（JSON）スキーマを使用して出力を検証し、高い精度を維持します。また、エッジケースを適切に処理し、下流システムと統合する必要があります。
システムは、アップロードされた履歴書から候補者の情報（氏名 連絡先、スキル、職務経歴、学歴）を抽出する必要があります。抽出されたデータは、事前に定義されたJSONスキーマに厳密に準拠する必要があり、必須項目が欠落していたり、データ型が間違っていたりすると、後続の検証でエラーが発生します。
クロードの出力がスキーマと常に一致することを保証するための最も確実な方法はどれですか？
A. 正規表現パターンを使用してクロードのテキスト応答を解析し、JSON オブジェクトを抽出します。不正な形式の応答には再試行ロジックを使用します。
B. システムプロンプトに詳細なJSONフォーマット手順とテンプレート例を含め、Claudeに有効なJSONのみを出力するように指示します。
C. 2 つの別々の API 呼び出しを実行します。まず情報をテキストとして抽出し、次に Claude にそのテキストを JSON 形式にフォーマットするように依頼します。
D. 必要な JSON 構造に一致する入力スキーマを持つツールを定義し、Claude の tool_use レスポンスからデータを抽出します。
Answer: (解答を表示する**)**
スキーマ定義ツールは、Claudeとアプリケーション間の構造化されたインターフェースを提供します。ツールのinput_schemaでは、必須プロパティ、ネストされた職務経歴および学歴オブジェクト、スキル配列、および正確なデータ型を宣言できます。Claudeは抽出した履歴書情報をツール呼び出し引数として提供し、アプリケーションはそれをtool_use応答から直接読み取ることができます。
Anthropic のツールドキュメントでは、カスタムツール定義には、期待されるパラメータを記述する JSON スキーマ オブジェクトが含まれると規定されています。現在の厳密なツール使用では、生成されたツール引数が宣言されたスキーマと完全に一致するように強制することもできます。これにより、自由形式のテキスト内の JSON を探したり、生成後の修復に依存したりする必要がなくなります。(https://docs.anthropic.com/en/docs/agents-and-tools/tool-use/implement- tool-use) オプション A は、正規表現が任意のネストされた JSON に対して堅牢なパーサーを提供せず、欠落したプロパティや誤った型を修正できないため、脆弱です。オプション B はフォーマット動作を改善しますが、確率的なままです。オプション C はレイテンシを 2 倍にし、抽出時と再フォーマット時に 2 つの情報損失の機会を生み出します。
実運用設計においては、履歴書に実際に記載されていない可能性のある項目は、架空の値を強制するのではなく、オプションまたはnull許容にするべきです。また、ツールスキーマでは、サポートされている場合はstrict: trueを使用する必要があります。利用可能な選択肢の中で、Dは最も強力な構造的保証と最もクリーンな下流統合を提供します。
公式の参考資料／トピック :ツール使用時の応答、入力スキーマ、厳密なツール使用、ネストされた構造化抽出。

最新問題: **34**
あなたはソフトウェア開発を加速するためにClaude Codeを使用しています。あなたのチームは、コード生成、リファクタリング、デバッグ、ドキュメント作成にClaude Codeを利用しています。カスタムスラッシュコマンドやCLAUDE.md

の設定を用いて開発ワークフローにClaude Codeを統合し、プランモードと直接実行の使い分けを理解する必要があります。
インフラストラクチャ -アズ -コードのリポジトリには、Terraform モジュール (/terraform/)、Kubernetes マニフェスト (/kubernetes/)、および CI/CD パイプライン スクリプト (/pipelines/)。それぞれ異なる規約が必要ですが、単一のルート CLAUDE.md は 500 行以上に膨れ上がっています。開発者が Kubernetes ファイルを操作すると、Terraform 固有のルールが不必要にコンテキストにロードされ、トークンが消費されます。
特定のファイルタイプを編集する際に、関連するガイダンスのみが表示されるように再編成するための最適な方法は何ですか？
A. YAMLフロントマターのパススコープ（例paths: ["terraform/**/*.tf"]）を使用して.claude/rules/にファイルを作成し、一致するファイルを編集するときにのみルールを読み込みます。
B. ルート CLAUDE.md を、ヘッダー (例: "## Terraform の規約") が付いた明確なラベルの付いたセクションに再構成し、構成と読みやすさを向上させます。
C. コンテンツをサブディレクトリ CLAUDE.md ファイル (/terraform/CLAUDE.md 、/kubernetes/CLAUDE.md) に分割します。
md) なので、Claude はディレクトリ固有のガイダンスを読み込みます。
D. ルート CLAUDE.md を保持し、@path/to/import 構文を使用して、別のドキュメントからツール固有のガイダンスファイルをモジュール化して含めます。
Answer: A (メーッセージを残す**)**
パススコープのルールは、Claudeが一致するファイルを扱う場合にのみ命令が読み込まれるという要件を直接満たします。各インフラストラクチャドメインは、.claude/rules/ の下に個別のMarkdownファイルを持つことができ、そのファイルにはTerraformファイル、Kubernetesマニフェスト、またはパイプライン構成を対象とするYAMLパスのフロントマターが含まれています。
Anthropic は、.claude/rules/ がプロジェクトのガイダンスをモジュール化し、グロブパターンに基づく条件付き読み込みをサポートすることを文書化しています。パス フィールドを持つルールは、Claude が一致するファイルを扱う場合にのみ適用され、無関係なコンテキストとトークンの消費を削減します。パス フロントマターのないルールは無条件に読み込まれます。(https://code.
claude.com/docs/en/memory)
オプションBは可読性を向上させますが、500行を超えるすべての行がセッションごとに残ってしまいます。オプションCはディレクトリ固有のCLAUDE.mdガイダンスを提供できますが、規約が複数のディレクトリにまたがる拡張子やパターンに依存する場合は、パススコープのルールの方がより正確です。オプションDは依然として手動でのインポートが必要であり、ドキュメントに記載されている条件付きルール読み込みメカニズムを提供しません。
適切な設計としては、terraform/**/*.tf をスコープとする terraform.md、kubernetes/**/*.{yaml,yml} をスコープとする kubernetes.md、パイプラインディレクトリをスコープとする pipelines.md などのルールを使用する。プロジェクト全体に適用される広範なコマンドとリポジトリのエチケットは、ルート CLAUDE.md に残しておくことができる。
公式の参照/トピック: .claude/rules/ ; パス固有のルール; YAML フロントマター; コンテキスト効率の良い命令の読み込み。

最新問題**: 35**

あなたはソフトウェア開発を加速するためにClaude Codeを使用しています。あなたのチームは、コード生成、リファクタリング、デバッグ、ドキュメント作成にClaude Codeを利用しています。カスタムスラッシュコマンドやCLAUDE.mdの設定を用いて開発ワークフローにClaude Codeを統合し、プランモードと直接実行の使い分けを理解する必要があります。
あなたは、データベーストランザクション、エラー処理、監査ログ記録に関するプロジェクトで確立されたパターンに従う必要がある、新しい決済処理モジュールを実装しようとしています。これらのパターンを例示する既存のモジュールとして、db_utils.py、error_handlers.py、audit_logger.py の 3 つを特定しました。これは一度限りの統合作業であり、これらのパターンはチームの Wiki に十分に文書化されているため、プロジェクトレベルでの追加ドキュメントは必要ありません。
最も効果的なアプローチは何ですか？
A. @参照を使用して、3つのモジュールをプロンプトに直接含め、クロードに具体的なコード例を示して、従うべきパターンを示します。
B. 各パターンのドキュメントをCLAUDE.mdファイルに追加し、それらをClaudeが自動的に適用するプロジェクト規約として確立します。
C. プロンプト内で、3つのモジュールのパターンを自然言語で説明し、Claudeが従うべきトランザクション処理方法、エラー形式、ログ記録規則について説明してください。
D. 新しいモジュールを生成する前に、Claude にコードベースを調べてもらい、トランザクション、エラー処理、ログ記録のパターンを見つけて理解してもらいます。
Answer: A (メーッセージを残す**)**
直接 @ 参照を使用すると、Claude は模倣する必要のある正確な実装を取得できます。人間が作成したドキュメントでは、@ を使用してファイルを参照すると、会話にファイルの内容全体が含まれ、1 つのメッセージで複数のファイルを参照できます。これにより、Claude はプロジェクトで使用される実際のトランザクション境界、例外構造、監査フィールド、命名規則、およびヘルパー API に即座にアクセスできます。(https://code.claude.com/docs/en
/共通ワークフロー）
オプションBは、タスクが明確に単発的なものであり、規約が既に他の場所で文書化されているため不適切です。CLAUDE.mdはすべてのセッションに読み込まれ、プロジェクト全体に広く適用できる簡潔な情報が含まれている必要があります。単一の統合のための詳細な実装資料を追加すると、コンテキストが不必要に消費されます。Anthropicは、時折発生する手順をスキルに移動し、CLAUDEはそのままにしておくことを推奨します。
md は、永続的で広く適用可能なガイダンスに限定されます。(https://code.claude.com/docs/en/memory) オプション C は、自然言語による要約では微妙だが重要なコードの動作が省略される可能性があるため、精度が低下します。
オプションDでは、クロードに既に特定されているファイルを再発見するように指示するため、探索時間とコンテキストの使用が増加します。
最も効果的なプロンプトは、3つのモジュールすべてを参照し、それぞれがどのパターンを示しているかを特定し、新しいモジュールに必要な動作を明記し、確立された慣例が遵守されたことを証明する集中的なテストを要求するものであるべきです。
公式の参照/トピック: @ ファイル参照、リッチプロンプトコンテキスト、CLAUDE.md スコープ、パターンベースの実装。

最新問題: **36**

あなたはソフトウェア開発を加速するためにClaude Codeを使用しています。あなたのチームは、コード生成、リファクタリング、デバッグ、ドキュメント作成にClaude Codeを利用しています。カスタムスラッシュコマンドやCLAUDE.mdの設定を用いて開発ワークフローにClaude Codeを統合し、プランモードと直接実行の使い分けを理解する必要があります。
あなたは、特定のパフォーマンス要件と、処理すべきエッジケース（切断されたノード、サイクル、重み付きエッジなど）を伴う複雑なグラフ探索アルゴリズムを実装しようとしています。Claude を使用して効率的な反復改良を行うためのワークフローを構築したいと考えています。
複数の反復作業を通して段階的な改善を最も効果的に実現できるアプローチは何でしょうか？
A. クロードにアルゴリズムを徹底的に調査させ、拡張思考を用いて詳細な実装計画を作成させ、その計画に基づいて完全なソリューションを実装させる。
B. ドキュメントから参照実装を Claude に提供し、コードベースのスタイルに合わせてコードを書き換え、必要なエッジケース処理を追加し、出力を参照と比較するように指示します。
C. 実装前に、期待される動作、エッジケース、およびパフォーマンス要件を網羅するテストスイートを作成します。クロードにテストに合格するコードを書いてもらい、テストの失敗を各改善要求と共有しながら反復作業を行います。
D. クロードに、すべての要件とエッジケースを含む、アルゴリズムの詳細な自然言語仕様を提供する。各出力を手動で確認し、変更が必要な動作について説明的なフィードバックを提供する。
Answer: C (メーッセージを残す**)**
オプションCは、客観的な検証ループを作成します。テストでは、非連結グラフ、サイクル処理、重み付きエッジ、無効な入力、およびパフォーマンス制約に対する期待される走査動作をエンコードします。クロードはアルゴリズムを実装し、テストスイートを実行し、具体的な不具合を検査し、測定可能な条件が満たされるまで実装を改良することができます。Anthropic は、テスト、ビルド、リンター、フィクスチャ比較などの検証メカニズムを Claude に与えることを重視しています。実行可能な合否判定チェックがない場合、Claude は実装が完了しているように見えることしか判断できません。テストがあれば、作業を実行し、結果を評価し、主観的な判断ではなく証拠に基づいて反復できます。Anthropic はまた、修正を適用する前に、失敗したテストで欠陥を再現することを推奨しています。(https://code.claude.com/docs/en/best-practices) オプション A は思慮深い初期設計を生み出す可能性がありますが、実装後の段階的な改善を保証するものではありません。オプション B は、プロジェクトの制約に合致しない可能性のある参照から仮定や欠陥を引き継ぐリスクがあります。オプション D は手動レビューに依存し、開発者を主要な検証システムに変えます。
テストスイートには、正当性検証フィクスチャ、境界条件、複雑性に敏感なワークロード、および新たな不具合が発見されるたびに追加される回帰テストを含める必要があります。これにより、すべての反復処理が累積的になります。つまり、修正は以前に検証された動作を損なうことなく、新たなケースを満たす必要があります。
公式の参考資料／トピック：実行可能ファイルの検証、テスト駆動型反復、フィードバックループ、回帰テスト。

最新問題**: 37**
あなたはソフトウェア開発を加速するためにClaude Codeを使用しています。あなたのチームは、コード生成、リファクタリング、デバッグ、ドキュメント作成にClaude Codeを利用しています。カスタムスラッシュコマンドやCLAUDE.mdの設定を用いて開発ワークフローにClaude Codeを統合し、プランモードと直接実行の使い分けを理解する必要があります。

アプリケーションにリアルタイム更新機能を追加することがあなたの任務です。この機能は、WebSocket、サーバー送信イベント、ポーリングなど、さまざまな方法で実装できますが、それぞれ複雑さ、ブラウザのサポート状況、インフラストラクチャ要件が異なります。
この作業を始める最も効果的な方法は何ですか？
A. まずは直接実行を使用してポーリングを実装し、その後、WebSocketsにアップグレードするかどうかを評価します。
B. クロードにすべてのアプローチを分析し、最適と判断したアプローチを実行するように指示するプロンプトを使用して直接実行します。
C. プランモードに入り、アーキテクチャを検討し、トレードオフを評価し、実装前にチームの承認を得るためのオプションを提示します。
D. WebSockets を使用した直接実行を開始し、インフラストラクチャの問題が発生した場合はリファクタリングします。
Answer: C (メーッセージを残す**)**
実装方法は未解決であり、アーキテクチャ上の影響を及ぼします。WebSocket、Server-Sent Events、ポーリングは、接続ライフサイクル、双方向通信、プロキシ互換性、スケーリング要件、デプロイメントトポロジー、再接続動作、運用上の複雑さにおいてそれぞれ異なります。したがって、クロードはコードを変更する前に、既存のアプリケーションとインフラストラクチャを調査する必要があります。
Anthropic は、アプローチが不確実な場合、作業が複数のファイルに影響を与える場合、または開発者が関連するアーキテクチャに精通していない場合は、探索と計画を実装から分離することを推奨しています。計画モードでは、Claude は変更を加えることなくファイルを読み取り、システムを分析して詳細な提案を作成し、開発者またはチームが実行前に計画を確認できるようにします。(https://code.claude.com/docs/en/best- practices) オプション A は、レイテンシと負荷特性が要件を満たしているかどうかを判断する前にポーリングを約束します。オプション B は、重要なアーキテクチャ上の決定と即時実装をレビュー チェック ポイントなしで単一の実行ステップに委任します。オプション D は、インフラストラクチャに最も敏感なオプションを時期尚早に選択し、後でのリファクタリングは安価であると想定します。
オプションCは正しい手順を示しています。すなわち、フロントエンドとバックエンドのアーキテクチャを検証し、要件を文書化し、代替案を比較し、インフラストラクチャの制約を特定し、承認を得てから、適切なテストと可観測性を備えた上で選択した設計を実装します。
公式参照資料／トピック：計画モード、探索 -計画 -実装ワークフロー、アーキテクチャ上のトレードオフ分析、レビューチェックポイント。

最新問題**: 38**
あなたはソフトウェア開発を加速するためにClaude Codeを使用しています。あなたのチームは、コード生成、リファクタリング、デバッグ、ドキュメント作成にClaude Codeを利用しています。カスタムスラッシュコマンドやCLAUDE.mdの設定を用いて開発ワークフローにClaude Codeを統合し、プランモードと直接実行の使い分けを理解する必要があります。
チームは、DevOpsワークフローテンプレートを提供するカスタムMCPサーバーを接続しました。このサーバーは、ツールに加えて、いくつかのMCPプロンプト（deploy_checklistやincident_responseなど）を公開します。
これらのMCPプロンプトは、Claude Code内でどのようにアクセスできるようになるのですか？
A. これらは追加のシステムレベルのコンテキストとしてすべての会話の先頭に自動的に追加され、セッション全体を通してクロードの行動に影響を与えます。

B. それらはサーバーのツールとともにClaude Codeのツールレジストリに追加され、タスクに関連する場合にモデルによって自動的に呼び出されます。
C. それらはファイルとともに @ メンション可能なリソースとして表示され、参照されると取得されてメッセージに添付されます。
D. それらはスラッシュコマンド（例/mcp__servername__deploy_checklist）として表示され、コマンド名の後に引数を渡して呼び出すことができます。
Answer: (解答を表示する**)**
MCPプロンプトは、自律的なツールや永続的にロードされるシステム命令ではなく、ユーザーが呼び出すコマンドとして公開されます。Claude Codeは、接続されているMCPサーバーからプロンプトを動的に検出し、/mcp__servername__promptnameという命名規則を使用してコマンドリストに表示します。
引数は、コマンドの後にスペースで区切られた値として指定されます。実行されると、MCP サーバーがプロンプトを解決し、返されたコンテンツがアクティブな会話に挿入されます。Anthropic の公式ドキュメントには、/mcp__github__list_prs や /mcp__jira__create_issue "ログイン フローのバグ" high などの例が示されています。(https://code.claude.com/docs/en/mcp) オプション A はコンテキストを継続的に消費し、オプションのワークフロー テンプレートを必須のシステム命令として誤って扱います。オプション B は、MCP プロンプトと MCP ツールを混同しています。ツールはモデル呼び出し可能な操作ですが、プロンプトはコマンドとして呼び出される再利用可能なプロンプト テンプレートです。オプション C は、参照および添付できますが、MCP の個別の機能である MCP リソースについて説明します。
指定されたサーバーの場合、チームは /mcp__devops__deploy_checklist などのコマンドを呼び出すことができます。/mcp__devops__incident_response service-name 。正確なサーバーセグメントは、設定されたサーバー名から取得され、必要に応じて正規化が適用されます。
公式リファレンス/トピック: MCP プロンプト、動的プロンプト検出、MCP スラッシュコマンド命名、プロンプト引数。

最新問題**: 39**
Claude Agent SDK を使用して、顧客サポート解決エージェントを構築しています。このエージェントは、返品、請求に関する紛争、アカウントの問題など、曖昧さの多いリクエストを処理します。カスタムのモデルコンテキストプロトコル (MCP) ツール (get_customer、lookup_order、process_refund、escalate_to_human) を介して、バックエンドシステムにアクセスできます。目標は、初回接触で 80% 以上の解決率を達成しつつ、エスカレーションが必要なタイミングを判断することです。
エージェントは、パスワードをリセットする前に、複数ステップのプロセスを経て顧客の本人確認を行います。テスト中に、顧客が3つ目の確認質問に答えた後、エージェントがまるで以前のやり取りがなかったかのように、顧客に再度名前を尋ねることに気づきました。
この行動の最も可能性の高い原因は何ですか？
A. プロンプトには、クロードに複数のやり取りにわたる情報を記憶するように指示する記述がありません。
B. 会話履歴が後続のAPIリクエストに渡されません。
C. 検証ツールは、検証ステップが成功するたびにエージェントの内部状態をクリアします。
D. クロードの記憶保持はデフォルトでは2回の会話ターンに制限されており、拡張するには明示的な設定が必要です。
Answer: B (メーッセージを残す**)**

Claude Messages APIはステートレスです。各APIリクエストには、Claudeが現在のレスポンスに必要な会話履歴を含める必要があります。アプリケーションが顧客の3回目の確認回答のみを送信した場合、Claudeは顧客の名前と以前の回答を含む以前のやり取りを自動的に取得しません。
モデルの観点からすると、その情報は存在しない。
Anthropic の Messages API ドキュメントには、アプリケーションはマルチターンインタラクションを構築するために会話履歴全体を送信する必要があると記載されています。正しい実装では、各ユーザーメッセージとアシスタントの応答をメッセージ配列に追加し、後続のリクエストごとに蓄積されたシーケンスを再送信します。(https://platform.claude.com/docs/en/build-with-claude/working-with-messages)
オプションAでは、リクエストから省略された情報を復元することはできません。 記憶する」指示では、サーバー側の会話状態は作成されません。オプションCは、シナリオでは説明されていないアプリケーション固有の動作について述べています。オプションDは事実と異なり、Claudeにはデフォルトの2ターン保持制限はありません。Claudeの有効な会話認識は、提供されるメッセージとモデルのコンテキストウィンドウに依存します。
検証ワークフローにおいては、アプリケーションは構造化された検証状態を、文章形式の履歴とは別に保持する必要があります。この状態には、完了したチェック、保留中のチェック、試行回数、および検証済みの顧客識別子などが含まれます。これにより信頼性が向上し、会話形式の記録は自然なやり取りを維持します。
公式の参考資料／トピック :ステートレスメッセージAPI、複数ターン履歴の構築、構造化されたワークフロー状態、会話の継続性。

最新問題: **40**
あなたはソフトウェア開発を加速するためにClaude Codeを使用しています。あなたのチームは、コード生成、リファクタリング、デバッグ、ドキュメント作成にClaude Codeを利用しています。カスタムスラッシュコマンドやCLAUDE.mdの設定を用いて開発ワークフローにClaude Codeを統合し、プランモードと直接実行の使い分けを理解する必要があります。
イベントの日付が未来の日付であることを確認する日付検証チェックを追加する必要があります。これには、単一ファイル内の既存の関数に条件文を追加する必要があります。
最も適切なアプローチは何でしょうか？
A. 直接実行を使用して変更を行います。
B. 検証ロジックについて徹底的に推論できるように、拡張思考モードを有効にして開始します。
C. 変更を行う前に、まず計画モードに入り、詳細な実施戦略を作成してください。
D. プランモードに入り、検証が予約フローの他の部分にどのような影響を与えるかを分析します。
Answer: (解答を表示する**)**
この変更は範囲が狭く、局所的で、既に定義済みです。つまり、単一ファイル内の既存の関数に条件付き検証チェックを1つ追加するだけです。別途計画段階を設けると、アーキテクチャ上の不確実性を解消することなく、プロセス上のオーバーヘッドが発生します。直接実行することで、クロードは関数を読み込み、条件を実装し、関連する集中的なテストを実行できます。
Anthropic は、計画モードはオーバーヘッドを追加するため、スコープが明確で修正が小さい場合は一般的にスキップすべきであると明示的に述べています。計画は、アプローチが不明確な場合、複数のファイルが影響を受ける場合、またはコードが不慣れな場合に最も価値があります。Anthropic の実践的なルールは、必要な差分を 1 文で説明できる場合は、直接実装が適切であるということです。(https://code.claude.com/docs/en/best-practices) オプション B は、単純な検証

ロジックに不要な推論の労力を割り当てています。オプション C と D は、単一機能の変更の複雑さを誇張しています。要件が予約のセマンティクス、タイムゾーンのルール、永続化の動作、またはパブリック インターフェイスを変更する場合にのみ、より広範な影響分析が正当化されますが、これらのいずれも明記されていません。
実装には検証を含めるべきです。クロードは、将来の日付、現在の日付、過去の日付に対するテストを追加または更新し、最も関連性の高いテストコマンドを実行する必要があります。直接実行は、検証されていない実行を意味するものではありません。
公式の参照資料／トピック：直接実行 計画モードの選択、小規模な範囲変更、集中的な検証。

最新問題**: 41**
あなたはソフトウェア開発を加速するためにClaude Codeを使用しています。あなたのチームは、コード生成、リファクタリング、デバッグ、ドキュメント作成にClaude Codeを利用しています。カスタムスラッシュコマンドやCLAUDE.mdの設定を用いて開発ワークフローにClaude Codeを統合し、プランモードと直接実行の使い分けを理解する必要があります。
チームのCLAUDE.mdには、「4スペースのインデントを使用し、常にPrettierフォーマットを実行する」というルールが含まれています。しかし、コードレビューの結果、Claude Codeが生成するファイルの約30%でフォーマットが不統一であることが判明しました。2スペースのインデントだったり、末尾のカンマが欠落していたりするケースが見られます。 重要：Prettierフォーマットを必ず使用してください」という強調を追加することで違反率は約15%に減少しますが、完全には解消されません。
生成されるすべてのコードが一貫したフォーマットになっていることを保証するための最も効果的な方法は何ですか？
A. 書式設定ルールを専用のスキルに抽出し、Claudeがコード生成時に自動的にロードするようにします。正しい書式設定のより詳細な例も含まれています。
B. 生成されたコードがフォーマット標準に準拠しているかどうかを評価し、違反があればクロードに修正を促すプロンプトベースのチェックを含む停止フックを追加します。
C. フォーマットルールをパススコープの .claude/rules/ ファイルに分割し、Claude が一致するファイルタイプを処理するときにロードします。
D. PostToolUse フックを Edit|Write マッチャーで設定し、Claude が変更する各ファイルに対して Prettier を自動的に実行します。
Answer: (解答を表示する**)**
書式設定は決定論的であり、より強力な自然言語による指示ではなく、決定論的なツールによって強制されるべきです。PostToolUse フックは、ツール操作が正常に完了した後に実行されます。Edit|Write に一致すると、フックはファイル変更に限定され、編集されたファイルパスを Prettier に直接渡すことができます。
Anthropic は、公式のフックガイダンスでこの正確なパターンを提供しています。PostToolUse イベントを Edit|Write マッチャーで構成し、変更されたファイルに対して Prettier を実行します。これにより、Claude がフォーマッタを実行することを覚えておくことに依存するのではなく、関連するすべての編集後にフォーマットが自動的に実行されることが保証されます。(https://code.
claude.com/docs/en/hooks-guide)
オプションAは依然として指示に従うことに依存しており、手続き的な要素を不必要にロードします。オプションBは、Prettierが結果を直接適用できるにもかかわらず、別の確率モデル評価によってフォーマットが正しいかどうかを判

断します。また、停止フックも必要以上に遅れて実行されます。オプションCはコンテキストの関連性を向上させますが、準拠を保証するものではありません。パススコープのルールは、適用制御ではなく指示のままです。
フックはプロジェクト設定に保存し、ワークフローをチーム全体で共有できるようにする必要があります。また、フォーマッタの終了ステータスも監視し、設定ミスや構文エラーが黙って無視されるのではなく、可視化されるようにする必要があります。
公式の参考資料/トピック: Claude Code Hooks; PostToolUse; Edit|Write Matchers; Deterministic Formatting Enforcement。

最新問題**: 42**
Claude Agent SDK を使用して、顧客サポート解決エージェントを構築しています。このエージェントは、返品、請求に関する紛争、アカウントの問題など、曖昧さの多いリクエストを処理します。カスタムのモデルコンテキストプロトコル (MCP) ツール (get_customer、lookup_order、process_refund、escalate_to_human) を介して、バックエンドシステムにアクセスできます。目標は、初回接触で 80% 以上の解決率を達成しつつ、エスカレーションが必要なタイミングを判断することです。
顧客は、最初のセッションから4時間後に、同じ請求に関する紛争について再び問い合わせてきました。前回の32ターンに及ぶセッションでは、lookup_orderの結果に「ステータス：保留中、解決予定：24〜48時間」と表示されていました。テストの結果、古いツール結果でセッションを再開すると、エージェントは、その後の新たなツール呼び出しで異なる情報が返された後でも、応答で古いデータを参照することが多いことがわかりました（例：「払い戻しはまだ処理中です」）。
リピーター顧客への対応において、最も確実な方法はどれか？
A. 履歴を完全に保持して再開し、セッション開始時に以前に使用したすべてのツールを自動的に再呼び出しするようにエージェントを設定して、データの鮮度を確保します。
B. 完全な履歴で再開し、同じツールへの複数の呼び出しがコンテキスト内に存在する場合、エージェントに常に最新のツール結果を優先するように指示するシステムプロンプト指示を追加します。
C. 履歴を完全に保持して再開しますが、再開する前に以前の tool_result メッセージをフィルタリングし、人間/アシスタントのターンのみを保持するため、エージェントは必要なデータを再取得する必要があります。
D. 新しいセッションを開始し、前回のやり取りの構造化された要約（問題の種類、実行されたアクション、解決状況）を挿入してから、新しいツール呼び出しを行ってから操作を開始します。
Answer: D (メーッセージを残す**)**
オプションDでは、永続的なケース履歴と変動の激しい運用データを分離します。新しいセッションには、請求に関する紛争、顧客の目的、以前に行われた措置、および未解決の状態を記述した、簡潔で構造化された要約が提供されます。古いバックエンドの観測データが依然として有効であるかのように継承されることはありません。
次に、エージェントが応答する前に、新しいツール呼び出しによって現在の払い戻しまたは注文の状態を取得します。
Agent SDK セッションは、以前のツール呼び出しやツール結果を含む会話履歴を保持します。そのため、完全なトランスクリプトを再開すると、外部バックエンドが 4 時間の間に大幅に変更されている可能性があるにもかかわらず、古いシステムデータがアクティブなコンテキストに再導入されます。会話の永続性と、外部システムの真実の永続性を混同しないでください。
オプションAは、再来店した顧客の現在の質問とは無関係なツールも含め、以前に使用したすべてのツールに対して不要な呼び出しを実行します。オプションBは、矛盾する過去の証拠を文脈に沿って保持しながら、迅速な対応を前提としています。オプションCは、既存のトランスクリプトからツールの結果を手動で削除するため、以前のtool_useブロックと

tool_resultブロック間の論理的な関係を損なう可能性がありますが、長くて構造化されていない会話はそのまま残ります。
構造化された要約には、安定した識別子、過去のアクション、顧客の約束、および未解決の問題が保持されるべきです。返金状況、配送状況、口座残高、解決予定など、時間的制約のあるフィールドは、信頼できるツールによって常に更新されるべきです。
公式参照資料／トピック :セッションの永続性、古いツール結果の管理、コンテキストの圧縮、最新データの取得。

最新問題: **43**
コードレビューのプロンプトには実装変更と対応するテストファイルの両方が含まれていますが、レビューコメントではテストされていないコードパスを特定できません。モデルはテストがまったくない関数を正しくフラグ付けしますが、テスト済みの関数内の条件分岐やエラー処理パスにカバレッジが不足していることを認識できません。パイプラインを複雑化せずに分岐レベルのギャップ検出を改善する最も効果的な方法は何ですか？
A. プロンプト内で実装とテストを交互に記述し、各関数をテストケースの直前に提示します。
B. クロードにすべての条件分岐と例外パスを列挙させ、各パスに対応するテストアサーションがあることを検証させる明示的な指示を追加します。
C. 2 パスのパイプラインを実装します。1 つのモデル呼び出しですべての条件分岐を抽出し、もう 1 つの呼び出しでそれらをテスト アサーションと相互参照します。
D. 未カバーの分岐を含むコードと、不足しているテストケースを特定する対応するレビューコメントを示す、いくつかのサンプルを含めます。
Answer: B (メーッセージを残す**)**
現在のプロンプトでは、テスト分析のレベルが高すぎます。クロードはテスト全体が明らかに欠落していることを認識していますが、パスレベルのインベントリを作成するように指示されていません。オプションBは、望ましい動作を明示的な検証手順に変換します。つまり、各条件、代替分岐、早期リターン、例外ハンドラ、および失敗パスを列挙し、その動作を実行するテストアサーションを見つけます。
Anthropicのプロンプトに関するベストプラクティスでは、明確な分析プロセスを必要とするタスクにおいては、明確で具体的な指示と明示的な手順を重視しています。今回の変更により、既存の単一レビュー呼び出しは維持しつつ、欠落していた評価基準を明確にすることができます。
オプション A は実装とテストの近接性を向上させますが、クロードにどのカバレッジ関係を検査すべきかを指示しません。オプション C は機能する可能性がありますが、直接的な指示が観測された障害を解決するかどうかをテストする前に、オーケストレーション、遅延、コスト、および別のハンドオフが追加されます。オプション D はデモに似た例の認識を向上させる可能性がありますが、いくつかのケースではすべての分岐構造を列挙できません。明示的なパス列挙は、馴染みのないコードにも一般化され、監査可能な出力を生成します。報告された各ギャップには、カバーされていない条件、期待される動作、および欠落しているアサーションを指定できます。

最新問題: **44**
Claude Agent SDK を使用して、開発者の生産性向上ツールを構築しています。このエージェントは、エンジニアが馴染みのないコードベースを探索したり、レガシーシステムを理解したり、定型コードを生成したり、反復作業を自動化したりするのに役立ちます。組み込みツールである Read、Write、Bash、Grep、Glob を使用し、Model Context Protocol (MCP) サーバーと統合します。

エンジニアは、本番ログに「SYNC_CONFLICT: エンティティバージョンの不一致が検出されました」という見慣れないエラーメッセージが表示されるのを確認しましたが、コードベース内の12個のサービスのうち、どれがそれを生成しているのか分かりません。そこでエンジニアは、原因となっているソースコードを特定するようエージェントに依頼しました。
責任のあるコードを最も効率的に見つけるには、どのような探索手法を用いるべきでしょうか？
A. grep を使用して、SYNC_CONFLICT や entity version mismatch などのエラー メッセージから特徴的なテキストを検索し、一致するファイルを読んでコンテキストを理解します。
B. Glob を使用して、errors、exceptions、handlers など、エラー処理によく関連付けられるディレクトリ内のファイルを検索し、一致するすべてのファイルを読み取ります。
C. プロジェクトのREADMEファイルとサービス設定ファイルを読み、その後、各サービスディレクトリ内のソースファイルを体系的に読みます。
D. grep を使用して、プロジェクトのエラー処理モジュールをインポートしているすべてのファイルを特定し、それらのファイルを読み込んでカスタムエラー定義を見つけます。
Answer: A (メーッセージを残す**)**
オプション A は、利用可能な最も強力な識別子である、正確な本番環境のエラー テキストから始まります。Anthropic のツール リファレンスでは、Grep がファイルの内容を検索して一致する行を返すと規定されており、エージェントはリポジトリ全体をロードすることなく、リテラル エラー定数またはメッセージを特定できます。安定コード SYNC_CONFLICT と特徴的なフレーズの両方を検索することで、識別子と表示されるメッセージが別々に定義されている場合を回避できます。エージェントは、一致するファイルのみを読み取り、周囲の呼び出しパス、エラー構造、およびサービス所有権を追跡できます。オプション B は、エラーが慣習的な名前のディレクトリに格納されていることを前提としていますが、12 個のサービスからなるリポジトリではこの前提が成り立たない場合があります。オプション C は、最も直接的な証拠を使用する前に、相当量のコンテキストを消費します。オプション D は、既知の共有エラー モジュールのコンシューマーに検索を絞り込みますが、メッセージはローカルで定義されているか、構成から生成されているか、または別のコンポーネントによってラップされている可能性があります。Grep に続いてターゲットを絞った Read 操作を実行することで、Anthropic のより広範なコンテキスト エンジニアリングの原則に従い、コード ベース全体をアクティブなコンテキストに配置するのではなく、必要なときに必要なソース マテリアルを取得します。

最新問題: **45**
Claude Agent SDK を使用して、開発者の生産性向上ツールを構築しています。このエージェントは、エンジニアが馴染みのないコードベースを探索したり、レガシーシステムを理解したり、定型コードを生成したり、反復作業を自動化したりするのに役立ちます。組み込みツールである Read、Write、Bash、Grep、Glob を使用し、Model Context Protocol (MCP) サーバーと統合します。
あなたはセキュリティスキャンワークフローを構築しています。
エンジニアが大規模なコードベース全体にわたって、eval() のような危険な関数のすべての出現箇所を特定する必要がある場合、エージェントはコンテンツ検索にどのツールを使用すべきでしょうか？
A. **/eval* のようなパターンで Glob を使用してファイルを探し、一致する各ファイルを読み取ります。
B. grep を使用して、コードベース内のすべてのファイルから正規表現パターン eval\(を検索します。
C. プロジェクトのメインエントリファイルを読み込み、import文をたどってeval()が使用されている可能性のある場所を追跡します。

D. Bash を使用して ls -R | grep eval を実行し、再帰的にリストされたファイル名を検索します。
Answer: B (メーッセージを残す)
オプションBは、ファイルの内容を検索するために設計されたツールを使用します。Anthropic社のClaude Codeツールリファレンスでは、GrepとGlobの違いについて説明しています。Grepはファイル内の行を検索するのに対し、Globはファイル名とパスを照合します。Grep は ripgrep をベースに構築されており、正規表現パターンを受け入れるため、関数呼び出しそのものに一致させたい場合は、開き括弧を eval\(のようにエスケープする必要があります。検索では、すべてのファイルをモデルのコンテキスト ウィンドウに読み込むことなく、一致するファイル、行番号、および周囲のコンテキストを返すことができます。オプション A は、eval に似たファイル名を検索し、その内容が関数を呼び出すソース ファイルを検索しません。オプション C は、選択したエントリ ポイントから到達可能なコードのみを追跡し、テスト ユーティリティ、動的にロードされるモジュール、スクリプト、および休止状態の脆弱なコードを見逃す可能性があります。オプション D は、再帰的なファイル名リストとテキスト フィルタリングを組み合わせたもので、ファイルの内容ではなく名前を検索します。Grep が一致を特定した後、エージェントは関連する範囲で Read を使用して、実際の実行可能呼び出しをコメント、文字列、または安全なラッパーから区別する必要があります。したがって、Grep は最も完全でコンテキスト効率の良い初期セキュリティ スキャンを提供します。

最新問題: **46**
実際の運用において、最終報告書には適切な出典表示のない主張が頻繁に含まれています。調査の結果、ウェブ検索エージェントと文書分析エージェントは出力に正しく引用を付加しているものの、統合エージェントは調査結果を組み合わせる際に、どの情報源がどの結論を裏付けているかを把握できていないことが判明しました。最も効果的なアーキテクチャの変更点は何でしょうか？
A. レポート生成器が元の情報源に対して意味的類似性マッチングを使用して主張の出所を再構築する検証ステップを追加します。
B. コーディネーターに、引き継ぎの前に文章中にソース識別子の接頭辞を挿入させ、レポート生成時にそれらの接頭辞を解析させる。
C. すべてのサブエージェントが、合成エージェントが結果を組み合わせる際に保持およびマージしなければならない構造化されたクレーム対ソースのマッピングを返すことを要求します。
D. すべてのサブエージェントのやり取りの完全な記録を保持し、レポート生成前にそれらのログを分析する引用解決エージェントを追加します。
Answer: C (メーッセージを残す)
オプションCでは、合成後に出所を再構築しようとするのではなく、データ契約の一部として出所を保持します。各発見事項には、安定したクレーム識別子と、URLまたは文書識別子、関連する場所、裏付けとなる抜粋、取得日、および該当する資格情報を含む1つ以上のソースレコードが必要です。
合成エージェントは、元の関係性を維持しながら、クレームを組み合わせたり書き換えたりすることができる。
Anthropicの構造化出力ドキュメントは、下流のワークフロー向けにスキーマ制約付きで解析可能な結果をサポートします。引用に関するドキュメントでは、信頼できる引用は提供されたソース資料への有効なポインタに依存すると説明しています。Anthropicのスキルガイダンスも同様に、完了前に各主要な主張を相互参照し、その引用を確認することを推奨しています。
オプションAは、帰属を確率的に再構築し、類似しているものの誤った記述と主張を関連付ける可能性があります。オプションBは、散文内にメタデータをエンコードするため、変換や解析が不安定になります。オプションDは、過剰なコンテ

キストを保持し、破棄されるべきではなかった情報を復元するために別のモデル段階を追加します。構造化された主張と情報源のマッピングにより、決定論的なマージ、引用の検証、重複排除、および監査証跡が可能になります。そのため、レポート生成ツールは、研究の完全な記録を再生することなく、各結論を裏付ける正確な証拠を引用できます。

最新問題: **47**
Claude Agent SDK を使用して、開発者向け生産性向上ツールを構築しています。このエージェントは、エンジニアが馴染みのないコードベースを探索したり、レガシーシステムを理解したり、定型コードを生成したり、反復作業を自動化したりするのに役立ちます。組み込みツール（読み取り、書き込み、Bash、Grep、Glob）を使用し、Model Context Protocol（MCP）サーバーと統合します。
開発者から、特定のAPIエンドポイントが断続的に500エラーを返す原因を調査するようエージェントに依頼がありました。コードベースには200以上のファイルがあり、開発者はどのコンポーネントが原因か把握していません。エージェントは、ルーティング、ミドルウェア、ビジネスロジック、データベース層を通してエラーを追跡する必要があります。
最も効果的なタスク分解手法はどれでしょうか？
A. エージェントに、ファイルの探索やコードの読み取りを開始する前に、エンドポイントを通るすべてのコードパスをマッピングする包括的な計画を最初に作成させます。
B. 事前に調査手順の固定シーケンスを定義します。エラーパターンをgrepで検索し、エラーハンドラを読み取り、データベースクエリを確認し、ミドルウェアを調査します。中間結果に関係なく、各手順を実行します。
C. 4つの層すべてを同時に調査する並列ワーカーエージェントを実行し、その結果を統合してエラーの発生源を特定します。
D. エージェントが各ステップで発見した内容に基づいて調査サブタスクを動的に生成し、エラーパスに関する新しい情報が出現するにつれて探索計画を適応させる。
Answer: D (メーッセージを残す**)**
原因となるファイル、コンポーネント、および実行シーケンスが不明なため、調査経路を確実に事前に決定することはできません。エージェントは、ルート定義、スタックトレース、ログ、エンドポイント参照などの利用可能な証拠から調査を開始し、それぞれの発見に基づいて次の検索、ファイル読み取り、または診断アクションを決定する必要があります。
Anthropicは、あらかじめ定義されたワークフローと、自身のプロセスとツールの使用を動的に制御するエージェントを区別します。エージェントは、必要なステップの数と性質を予測したり、固定パスとしてエンコードしたりできない、オープンエンドの問題に適しています。実行中、エージェントはツールの結果からグラウンドトゥルースを取得し、その環境フィードバックに基づいて計画を調整する必要があります。（https://www.anthropic.com）/engineering/building-effective-agents ）
オプションAでは、エージェントがコードを検査する前に包括的な計画が必要となるため、その計画は根拠のない仮定に基づくことになります。オプションBでは、初期の発見によって後のステップが無意味になったり、異なる依存関係パスが特定されたりした場合でも、すべての調査を同じ順序で進めることを強制します。オプションCでは、4つのレイヤーを

独立して調査できると想定していますが、断続的なリクエストの失敗を追跡するには、通常、レイヤー間で順次明らかになる依存関係をたどる必要があります。
オプションDは、適応型エージェントループを実装します。具体的には、調査、仮説の形成、ツールの使用、証拠の評価、そして次のサブタスクの生成という流れです。ワークフローには、停止条件、検証可能な仮説、そして証拠が決定的な結論に至らない場合のエスカレーションも含まれるべきです。
公式参考文献／トピック：適応型エージェントループ、動的タスク分解、ツールフィードバック、自由記述式コーディング調査。

最新問題: **48**
Claude Agent SDK を使用して、開発者向け生産性向上ツールを構築しています。このエージェントは、エンジニアが馴染みのないコードベースを探索したり、レガシーシステムを理解したり、定型コードを生成したり、反復作業を自動化したりするのに役立ちます。組み込みツール（読み取り、書き込み、Bash、Grep、Glob）を使用し、Model Context Protocol（MCP）サーバーと統合します。
昨日、あるエンジニアがエージェントを使って既存の認証モジュールを分析し、マイクロサービス抽出とインプレースリファクタリングという2つの異なるリファクタリング手法を特定しました。今日、彼らはどちらの手法を採用するかを決定する前に、エージェントにそれぞれの手法について具体的なコード変更を提案させ、両方の手法を詳細に検討したいと考えています。
この調査を最も効果的に構成するにはどうすればよいでしょうか？
A. 新たに2つのセッションを開始し、昨日の分析結果の要約を手動で入力してコンテキストを確立します。
B. 昨日のセッションを再開し、同じ会話スレッド内で両方のアプローチを順番に検討します。
C. 昨日の分析から fork_session を使用して 2 つのブランチを作成し、それぞれのフォークで 1 つのアプローチを検証します。
D. 昨日のセッションを再開して最初のアプローチを検証し、次に新しいセッションを開始して2番目のアプローチを検証し、元のコンテキストを手動で再現します。
Answer: (解答を表示する**)**

最新問題: **49**
25回以上も請求に関する紛争を調査した結果、重複請求は決済ゲートウェイのタイムアウトによって再試行ロジックがトリガーされたことが原因であることが判明しました。必要な返金額847ドルは、承認限度額500ドルを超えているため、escalate_to_humanを呼び出す必要があります。人間のエージェントは会話の記録にアクセスできません。効果的な解決を可能にするために、どのようなコンテキストを渡すべきですか？
A. すべてのメッセージとツール結果を含む完全な会話記録。
B. 顧客からの苦情の原文と、重複取引を示すツール結果の抜粋。
C. 顧客識別子、検証済みの根本原因、返金額、関連する取引識別子、既に試みたアクション、推奨される次のアクションを含む構造化された要約。
D. 診断と返金金額のみ。
Answer: C (メーッセージを残す**)**
オプションCでは、長時間の会話を再生することなく処理を続行するために必要な運用状態をエージェントに提供します。引き継ぎには、検証済みの識別子、重複トランザクションの証拠、診断されたタイムアウトおよび再試行メカニズ

ム、847ドルの払い戻し要件、エージェントの500ドルの承認制約、完了した検証手順、以前のアクション、および推奨される解決策を含める必要があります。未解決の不確実性はすべて明示的にラベル付けする必要があります。
Anthropicの効果的なコンテキストエンジニアリングに関するガイダンスでは、冗長な会話履歴やツール呼び出し履歴を削除しつつ、価値の高い状態を構造化されたメモに保持することを推奨しています。また、同社の長期稼働エージェントに関するガイダンスでは、コンテキストのリセットやエージェントの境界を越えて継続性を維持するためのメカニズムとして、構造化されたハンドオフについて説明しています。
オプションAは生の情報を最大限に活用しますが、人間が25回以上もやり取りする中で関連する事実を探し出す必要があり、遅延とエラーのリスクが高まります。オプションBは証拠を保持しますが、承認制約、完了した検証、および明示的な推奨アクションが省略されます。オプションDは情報が少なすぎて、検証や実行をサポートできません。構造化されたハンドオフは、忠実性と効率性のバランスが取れています。次の担当者が払い戻しの決定を下すために必要なすべてを含みながら、挨拶、繰り返しの説明、および無関係な中間ツールの出力は除外します。

最新問題: **50**
あなたはClaudeを使用して構造化データ抽出システムを構築しています。このシステムは、非構造化ドキュメントから情報を抽出し、JavaScript Object Notation（JSON）スキーマを使用して出力を検証し、高い精度を維持します。また、エッジケースを適切に処理し、下流システムと統合する必要があります。
テストの結果、ソースドキュメントに特定の仕様が欠落している場合、モデルはスキーマの必須フィールドを満たすためにもっともらしい値を捏造することが判明しました。たとえば、寸法のみが記載されているドキュメントの場合、抽出出力には「重量：2.3 kg」という捏造された値が表示されます。
この幻覚行動に最も効果的に対処できるスキーマ設計変更はどれですか？
A. プロンプトに「文書に明示的に記載されている情報のみを抽出し、欠落値にはプレースホルダーテキストを使用する」という明確な指示を追加する。
B. ソースドキュメントに存在しない可能性のあるフィールドを必須からオプションに変更し、モデルがそれらを省略できるようにします。
C. モデルが自身の確信度を自己報告する各仕様の横に「信頼度」フィールドを追加し、信頼度の低い抽出をフィルタリングします。
D. 抽出された各値がソース文書のテキスト中に存在するか、またはソース文書のテキストから推測できることを検証する意味検証を実装します。
Answer: B (メーッセージを残す**)**
このスキーマは、捏造を誘発する構造的なインセンティブを生み出しています。フィールドが必須と宣言されている場合、ソース ドキュメントに対応する証拠がない場合でも、出力には値を含める必要があります。構造化出力は、Claude の応答が JSON スキーマに準拠することを保証できますが、スキーマへの準拠は、生成されたすべての値が事実に基づいていることを保証するものではありません。Anthropic のドキュメントには、必須配列によってどのプロパティが存在する必要があるかが決まることがわかります。したがって、正当に存在しない可能性のあるソース依存のプロパティは、必須フィールドとして含めるべきではありません。(https://platform.claude.com/docs/en/build- with-claude/structured-outputs) オプション B は、契約レベルで問題を修正します。Claude は、有効な JSON を生成するためだけにコンテンツを捏造するのではなく、利用できないプロパティを省略できます。下流のシステムが安定したキー セットを必要とする場合は、null 許容表現を使用することもできますが、サポートされていない非 null 値を強制することは、アーキテクチャ的に健全ではありません。

オプションAは依然としてプレースホルダーの生成を必要とし、スキーマと利用可能な証拠との不一致を解消しません。
オプションCはモデルによって生成された信頼度に依存していますが、これはグラウンディングの代わりにはなりません。
オプションDは有用な補助制御手段ではあるが、要求されたスキーマ設計の変更には該当しない。
Anthropic は、不確実性を許容し、事実に基づく主張は提供された資料に基づいている必要があることを推奨しています。（https://docs.anthropic.com/en/docs/test-and-evaluate/strengthen-guardrails/reduce-hallucinations）公式の参照/トピック：構造化出力 JSON スキーマ設計、幻覚の削減 - 不確実性の許容と主張の根拠付け。

最新問題: **51**
Claude Agent SDK を使用して、顧客サポート解決エージェントを構築しています。このエージェントは、返品、請求に関する紛争、アカウントの問題など、曖昧さの多いリクエストを処理します。カスタムのモデルコンテキストプロトコル (MCP) ツール (get_customer、lookup_order、process_refund、escalate_to_human) を介して、バックエンドシステムにアクセスできます。目標は、初回接触で 80% 以上の解決率を達成しつつ、エスカレーションが必要なタイミングを判断することです。
エージェントの MCP ツールに配送固有の機能 (check_delivery_status、contact_driver、issue_credit、apply_promo_code、update_delivery_address、reschedule_delivery) を追加した後、ツールの総数は 4 から 10 に増加しました。評価スイートでは、ツールの選択精度が 88% から低下していることが示されています。
71%。ログ分析によると、エラーの大部分は、エージェントが意味的に重複するツールを選択する際に発生するもので、process_refundが正しい場合にissue_creditを呼び出したり、lookup_orderが既に必要なデータを返しているにもかかわらずcheck_delivery_statusを呼び出したりしている。
ログにエラーの原因として特定された意味的な重複を構造的に排除するには、どの手法が有効でしょうか？
A. ツールを2つのサブエージェントに分割します。1つはprocess_refund、issue_credit、apply_promo_codeを実行する「財務解決」エージェント、もう1つは残りの配送ツールを実行する「配送業務」エージェントで、コーディネーターがそれらの間をルーティングします。
B. 意味的に重複するツールを統合します。issue_credit と process_refund をアクション パラメータを持つ単一の resolve_compensation ツールに統合し、check_delivery_status をオプションの include_tracking フラグを持つ lookup_order に統合します。
C. 6 つの新しいツールに対して defer_loading を使用してツール検索ツールを有効にし、元の 4 つのツールは常にロードされたままにしておくことで、エージェントは必要なときにのみ特殊ツールを動的に検出します。
D. システムプロンプトに、曖昧なツールペアごとに正しい選択を示すいくつかの例を追加します。たとえば、issue_credit が適用される場合と process_refund が適切な場合を示します。
Answer: (解答を表示する**)**
オプションBは、ツールインターフェース自体から曖昧さを解消します。エージェントは、密接に関連する業務操作を表す別々の機能を選択する必要がなくなります。代わりに、単一の報酬ツールが明確なアクションパラメータを公開し、既存の注文検索ツールは、明確に定義されたフラグを通じて、オプションで配送追跡情報を返すことができます。
Anthropicのツール設計ガイドラインでは、個別の機能が不必要な意味的重複を生み出す場合、関連する操作をより少なく、より高性能なツールに統合することを推奨しています。Claudeは主にツールの名前、説明、スキーマ、および現在のタ

スクに基づいてツールを選択します。複数のツールが同じ要求を満たすことができるように見える場合、エージェントはインターフェース設計で明示的に示されるべき区別を推測する必要があるため、選択精度が低下します。
オプションAはルーティングの複雑さを増しますが、同じサブエージェント内で重複する金融操作が引き続き利用可能になります。オプションCは初期ロードされるツールの数を減らしますが、補償操作間または重複する検索機能間の意味的な重複を解消しません。オプションDは例を通して動作を改善する可能性がありますが、根本原因を修正するのではなく、設計の不十分なツールインターフェースを補うものです。
統合されたスキーマでは、サポートされるアクションには列挙型を使用し、各アクションが適用されるタイミングを明確に定義し、返されるフィールドを文書化する必要があります。これにより、エージェントとコンピュータ間のインターフェースがより小さく、より決定論的になります。
公式参照/トピック: ツールの統合、セマンティックツールの境界、アクションパラメータ、MCPツールの選択精度。

最新問題: **52**
Claude Agent SDK を使用して、開発者向け生産性向上ツールを構築しています。このエージェントは、エンジニアが馴染みのないコードベースを探索したり、レガシーシステムを理解したり、定型コードを生成したり、反復作業を自動化したりするのに役立ちます。組み込みツール（読み取り、書き込み、Bash、Grep、Glob）を使用し、Model Context Protocol（MCP）サーバーと統合します。
昨日、あるエンジニアがエージェントを使って既存の認証モジュールを分析し、マイクロサービス抽出とインプレースリファクタリングという2つの異なるリファクタリング手法を特定しました。今日、彼らはどちらの手法を採用するかを決定する前に、エージェントにそれぞれの手法について具体的なコード変更を提案させ、両方の手法を詳細に検討したいと考えています。
この調査を最も効果的に構成するにはどうすればよいでしょうか？
A. 昨日の分析から fork_session を使用して 2 つのブランチを作成し、それぞれのフォークで 1 つのアプローチを検証します。
B. 昨日のセッションを再開し、同じ会話スレッド内で両方のアプローチを順番に検討します。
C. 昨日のセッションを再開して最初のアプローチを検証し、次に新しいセッションを開始して2番目のアプローチを検証し、元のコンテキストを手動で再現します。
D. 新たに2つのセッションを開始し、昨日の分析結果の要約を手動で入力してコンテキストを確立します。
Answer: A (メーッセージを残す)
フォーク機能は、共有された過去の分析結果から代替的な方向性を探るために特別に設計されています。各フォークは、読み込まれたファイル、アーキテクチャ上の観察結果、依存関係の発見、既に記録された決定事項など、昨日の会話履歴のコピーから始まります。マイクロサービスアプローチとインプレースリファクタリングアプローチは、それぞれ別のセッションIDの下で独立して開発できます。
Anthropic の Agent SDK ドキュメントには、フォークは元の履歴のコピーから新しいセッションを作成し、元のセッションは変更されないと記載されています。結果として得られる各セッションは、その後独立して再開できます。ドキュメントに記載されている実装では、Python では resume と fork_session=True、TypeScript では forkSession: true を組み合わせています。(https://code.claude.com/docs/en/agent-sdk/sessions) オプション B では、最初のアプローチからの結論、仮定、および提案された編集が、2 番目の評価に影響を与える可能性があります。オプション C では、1 つのブランチのコンテキストのみが保持され、エンジニアは他のブランチのコンテキストを手動で再構築する必要があります。オプション D では、セッションですでにキャプチャされた詳細な分析が破棄され、不完全な要約に依存する可能性があります。

2つのフォークは同等の開始条件を提供し、親の調査を維持し、範囲、移行リスク、運用上の複雑さ、および必要なコード変更の公平な比較をサポートします。セッションのフォークは作業ディレクトリではなく会話履歴を分岐させるため、ファイルシステムの編集はワークツリーまたはチェックポイントによって分離する必要があります。
公式リファレンス/トピック: エージェントSDKセッション、セッションフォーク、代替アプローチの探索、コンテキストの保持。

最新問題: **53**
Claude Agent SDK を使用して、顧客サポート解決エージェントを構築しています。このエージェントは、返品、請求に関する紛争、アカウントの問題など、曖昧さの多いリクエストを処理します。カスタムのモデルコンテキストプロトコル (MCP) ツール (get_customer、lookup_order、process_refund、escalate_to_human) を介して、バックエンドシステムにアクセスできます。目標は、初回接触で 80% 以上の解決率を達成しつつ、エスカレーションが必要なタイミングを判断することです。
process_refund ツールは、2 種類のエラーを返します: 技術的なエラー ("503 Service Unavailable",
一時的なエラー（通話の約5%）と、永続的なビジネスエラー（「注文が30日間の返品期間を超えています」、「商品は既に返金されています」）（通話の約12%）があります。モニタリングによると、エージェントは3〜
4ターンにわたって、決して成功しないビジネスエラーを再試行します。現在、どちらのエラータイプもクロードにはプレーンテキストメッセージのみを返します。
顧客対応の質を向上させつつ、無駄な再試行を減らすための最も効果的な方法は何ですか？
A. 技術的なエラーに対してのみツール層で自動再試行ロジックを実装し、ビジネスエラーは再試行せずにClaudeに渡します。
B. エラーメッセージのテキストを解析して、再試行可能なエラーと再試行不可能なエラーを区別する方法を示すいくつかの例を追加します。
C. ビジネスルール違反を防ぐため、process_refundの前に呼び出す必要があるcheck_refund_eligibilityツールを追加します。
D. ビジネスエラーの場合は「retriable」: false を指定して構造化されたエラー応答を返し、Claude が使用できる顧客向けの説明を返してください。
Answer: D (メーッセージを残す**)**
エラー応答では、制御セマンティクスとユーザー向けの意味の両方を明確に伝える必要があります。「retriable」をfalseに設定すると、同じ操作を繰り返しても結果が変わらないことをエージェントに伝えます。顧客にとって分かりやすい説明を提供することで、Claudeは内部実装の詳細を公開したり、ビジネスルールを独自に解釈したりすることなく、正確に応答できます。
Anthropic は、失敗したツール操作をエラー インジケータと十分な情報を含むコンテンツとともに返すことを推奨しています。これにより、Claude は再試行するか、別の入力を要求するか、制限を説明するかを判断できます。ツール インターフェイスは、Claude に曖昧なテキストから操作動作を推測させるのではなく、詳細な説明と構造化されたパラメータを提供する必要があります。(https://platform.claude.com/docs/en/agents-and-tools/tool-use/build-a-tool- using-agent?utm_source=chatgpt.com) オプション A は、自動再試行を技術的な障害に適切に制限しますが、ビジネス上の障害にも明確な意味がない限り、述べられている顧客応答の問題を解決しません。オプション B は、人間が読めるメッセージの脆弱な解析に依存しています。オプション C は、遅延と別のツールへの依存を追加し、事前チェックでカバーされていない理由で適格性が変更または失敗する可能性があります。

完全なスキーマでは、エラーコード、カテゴリ、再試行可能性、顧客にとって安全な説明、および許可される次のアクションを特定する必要があります。技術的なエラーの場合も同様に、「再試行可能」: true が返され、再試行に関するガイダンスと最大試行回数ポリシーが示されます。
公式参照資料／トピック：構造化ツールエラー、再試行可能性の分類、顧客にとって安全な説明、回復力のあるMCP契約。