

生成AIアプリ開発者のワークフロー

堅牢な生成AI アプリケーション (生成AI app) を開発するには、綿密な計画、迅速な開発 - フィードバック - 評価ループ、スケーラブルな本番運用インフラストラクチャが必要です。このワークフローでは、最初のご概念実証 (POC) から本番運用のデプロイまでをガイドする推奨される一連の手順の概要を示します。

0. 前提条件

- 要件を収集して、生成AIの適合を検証し、制約を特定します。
- ソリューション アーキテクチャを設計します。[エージェント システムの設計パターン](#)を参照してください

1. ビルド

- データソースを準備し、必要なツールを作成します。
- 初期プロトタイプ (POC) をビルドして検証します。
- 本番運用前の環境にデプロイします。

2. 評価と反復

- ユーザーフィードバックの収集と品質の測定
- 評価に基づいてエージェントのロジックとツールを改良することにより、品質の問題を修正します。
- SME (Subject Matter Expert) の意見を取り入れて、エージェントシステムの品質を継続的に改善します。

3. 本番運用

- 生成AI アプリを本番運用環境にデプロイします。
- パフォーマンスと品質を監視します。
- 実際の使用状況に基づいて維持および改善します。

このワークフローは、デプロイまたは評価サイクルのたびに、前の手順に戻ってデータパイプラインを改良したり、エージェントロジックを更新したりする反復的なものにする必要があります。たとえば、本番運用 モニタリングでは、新しい要件が明らかになり、エージェントの設計が更新され、別の評価が行われる可能性があります。これらの手順を体系的に実行し、Databricks MLflow トレース、エージェントフレームワーク、エージェント評価機能を活用することで、ユーザーのニーズを確実に満たし、セキュリティとコンプライアンスの要件を尊重し、時間の経過とともに改善を続ける高品質の生成 AI アプリを構築できます。

0. 前提条件

生成AIアプリケーションの開発を始める前に、要件の収集とソリューション設計を適切に行うために時間をかけることがいかに重要であるかは、いくら強調してもし過ぎることはありません。

[要件の収集](#) には、次の手順が含まれます。

- 生成AIがお客様のユースケースに適合するかどうかを検証します。
- ユーザーエクスペリエンスを定義する。
- データソースの範囲を絞り込みます。
- パフォーマンスの制約を設定します。
- セキュリティ制約をキャプチャします。

[ソリューション設計](#) には、次のものが含まれます。

- データパイプラインをマッピングします。
- 必要なツールを特定します。
- 全体的なシステムアーキテクチャの概要を説明します。

この土台を築くことで、後続の [構築](#)、[評価](#)、および [本番運用](#) の各ステージの明確な方向性を設定します。

要件の収集

明確で包括的なユースケース要件を定義することは、成功する生成AIアプリを開発するための重要な最初のステップです。これらの要件は、次の目的に役立ちます。

- これらは、生成AIアプローチがユースケースに適しているかどうかを判断するのに役立ちます。
- これらは、ソリューションの設計、実装、評価の決定を導きます。

最初に時間をかけて詳細な要件を収集することで、開発プロセスの後半で大きな課題が発生するのを防ぎ、結果として得られるソリューションがエンドユーザーや利害関係者のニーズを満たすことを確認できます。明確に定義された要件は、アプリケーションのライフサイクルの後続の段階の基盤を提供します。

そのユースケースは生成AIに適していますか？

生成AIソリューションにコミットする前に、その固有の強みが要件と一致しているかどうかを検討してください。生成AIソリューションが適している例には、次のようなものがあります。

- **コンテンツ生成:** このタスクでは、静的テンプレートや単純なルールベースのロジックでは実現できない斬新なコンテンツや創造的なコンテンツを生成する必要があります。

- **動的クエリ処理:** ユーザークエリは自由回答形式または複雑で、柔軟でコンテキストに応じた応答が必要です。
- **情報合成:** このユースケースでは、多様なソース of 情報を組み合わせて要約することで、一貫性のある出力を生成することができます。
- **エージェントシステム:** アプリケーションに必要なのは、プロンプトに応答してテキストを生成するだけではありません。次の機能が必要な場合があります。
 - **計画と意思決定:** 特定の目標を達成するための多段階の戦略を策定する。
 - **アクションの実行:** 外部プロセスをトリガーしたり、さまざまなツールを呼び出してタスク (データの取得、API 呼び出しの実行、SQL クエリの実行、コードの実行など) を実行したりします。
 - **状態の維持:** 複数のインタラクションにわたる会話履歴やタスクのコンテキストを追跡して、継続性を確保します。
 - **適応型出力の生成:** 以前のアクション、更新された情報、または変化する条件に基づいて進化する応答を生成します。



逆に、生成AIのアプローチは、次のような状況では理想的ではないかもしれません。

- このタスクは非常に決定論的であり、事前定義されたテンプレートまたはルールベースのシステムを使用して効果的に解決できます。
- 必要な情報のセット全体は、すでに静的であるか、シンプルで閉じたフレームワークに収まっています。
- 非常に低いレイテンシ(ミリ秒)の応答が必要であり、生成処理のオーバーヘッドには対応できません。
- 目的のユースケースには、単純なテンプレート化された応答で十分です。

❗ 備考

PO、P1、および P2 ラベルに関する注意


以下のセクションでは、ラベル **PO**、**P1**、**P2** ラベルを使用して相対的な優先度を示します。

-  **[PO]** アイテムは重要または不可欠です。これらはすぐに対処する必要があります。
-  **[P1]** アイテムは重要ですが、**PO** 要件の後に従うことができます。
- **(2)** 項目は、時間とリソースが許す限り対処できる優先度の低い考慮事項または機能強化です。

これらのラベルは、チームがすぐに対応が必要な要件と延期できる要件をすばやく確認するのに役立ちます。

ユーザーエクスペリエンス

ユーザーが生成AIアプリとどのように対話し、どのような応答が期待されるかを定義します。

-  **[PO]典型的なリクエスト:** 一般的なユーザーリクエストはどのようなものですか?利害関係者から例を収集します。

- **[P0] 期待される回答:** システムはどのような種類の応答を生成する必要がありますか (たとえば、短い回答、長い形式の説明、創造的な説明など)?
- **[P1] インタラクションモダリティ:** ユーザーはアプリケーションとどのように対話しますか (チャット インターフェイス、検索バー、音声アシスタントなど)?
- **[P1] トーン、スタイル、構造:** 生成された出力は、どのようなトーン、スタイル、および構造 (形式的、会話的、技術的、箇条書き、または連続的な散文) を採用する必要がありますか?
- **[P1] エラー処理:** アプリケーションは、あいまいなクエリ、不完全なクエリ、またはターゲットから外れたクエリをどのように処理する必要がありますか? フィードバックを提供するべきか、それとも説明を求めるべきか?
- **[P2] フォーマット要件:** 出力 (メタデータや補足情報を含む) に関する特定の書式設定または表示ガイドラインはありますか?

データ

生成AI アプリで使用されるデータの性質、ソース、品質を決定します。

- **[P0] データソース:** どのようなデータソースがありますか?
 - 各ソースについて、以下を決定します。
 - データは構造化されていますか、それとも構造化されていませんか?
 - ソース形式 (PDF、HTML、JSON、XML など) は何ですか?
 - データはどこにありますか?
 - 利用可能なデータの量はどれくらいですか?
 - データにどのようにアクセスする必要がありますか?
- **[P1] データ更新:** データはどのくらいの頻度で更新されますか?更新を処理するためにどのようなメカニズムがありますか?
- **[P1] データ品質:** 既知の品質問題や不整合はありますか?
 - データソースに対する質の高いモニタリングが必要かどうかを検討してください。

次の例で、この情報を統合するためにインベントリ テーブルを作成することを検討してください。

データソース	ソース	ファイルの種類	サイズ	更新頻度
データソース 1	Unity Catalog ボリューム	JSON	10 GB	日次

データソース	ソース	ファイルの種類	サイズ	更新頻度
データソース 2	公開API	XML	NA (API)	リアルタイム
データソース 3	SharePoint	PDFファイル、.docx	500メガバイト	毎月

パフォーマンスの制約

生成AIアプリケーションのパフォーマンスとリソース要件をキャプチャします。

レイテンシー

- **[PO] 最初のトークンまでの時間:** 出力の最初のトークンを配信する前に許容される最大遅延はどれくらいですか?
 - 注意：レイテンシは通常、p50 (中央値) と p95 (95 パーセンタイル) を使用して測定され、平均パフォーマンスとワーストケースのパフォーマンスの両方をキャプチャします。
- **[PO] 完了までの時間:** ユーザーにとって許容できる (完了までの時間) 応答時間はどれくらいですか?
- **[PO] ストリーミング レイテンシ:** レスポンスがストリームの場合、全体的なレイテンシを高くしても許容されますか?

スケーラビリティ

- **[P1] 並列ユーザーおよびリクエスト:** システムは何人の同時ユーザーまたはリクエストをサポートする必要がありますか?
 - 注: スケーラビリティは、多くの場合、QPS (クエリ/秒) または QPM (クエリ/分) で測定されます。
- **[P1] 使用パターン:** 予想されるトラフィック パターン、ピーク負荷、または時間ベースの使用量の急増は何か?

コストの制約

- **[PO] 推論コストの制限:** 推論コンピュート リソースのコスト制約または予算制限は何ですか?

評価

生成AIアプリがどのように評価され、時間の経過とともに改善されるかを確立します。

- **[PO] ビジネス KPI**: アプリケーションが影響を与えるべきビジネス目標または KPI は何か? ベースライン値と目標を定義します。
- **[PO] ステークホルダーフィードバック**: アプリケーションのパフォーマンスとアウトプットに関する初期および継続的なフィードバックを提供するのは誰ですか? 特定のユーザーグループまたはドメインエキスパートを特定します。
- **[PO] 品質の測定**: 生成された出力の品質を評価するために、どのようなメトリクス (精度、関連性、安全性、ヒューマンスコアなど) が使用されますか?
 - これらのメトリクスは、開発中にどのようにコンピュータされますか (たとえば、合成データに対して、手動でキュレーションされたデータセットに対して)?
 - 本番運用では、品質はどのように測定されますか (たとえば、実際のユーザーからの問い合わせに対する応答のログ記録と分析)?
 - エラーに対する全体的な許容度はどの程度ですか? (たとえば、事実の軽微な不正確さを一定の割合で受け入れたり、重要なユースケースに対してほぼ100%の正確性を要求したりします。
 - その目的は、実際のユーザークエリ、合成データ、またはその両方の組み合わせから評価セットに向けて構築することです。このセットは、システムの進化に合わせてパフォーマンスを評価するための一貫した方法を提供します。
- **[P1] フィードバックループ**: ユーザーフィードバック (親指を立てる/下げる、アンケートフォームなど) をどのように収集し、反復的な改善を推進するために使用するべきか?
 - フィードバックをレビューして反映する頻度を計画します。

安全

セキュリティとプライバシーに関する考慮事項を特定します。

- **[PO] データの機密性**: 特別な取り扱いが必要な機密データ要素はありますか?
- **[P1] アクセス制御**: 特定のデータや機能を制限するためにアクセス制御を実装する必要がありますか?
- **[P1] 脅威の評価と軽減**: アプリケーションは、プロンプトインジェクションや悪意のあるユーザー入力など、一般的な生成AIの脅威から保護する必要がありますか?

配備

生成AIソリューションがどのように統合、デプロイ、監視、保守されるかを理解します。

- **[P1] 統合**: 生成AIソリューションは、既存のシステムやワークフローとどのように統合すべきか?
 - 統合ポイント (Slack、CRM、BI ツールなど) と必要なデータ コネクタを特定します。

- 生成 AI アプリとダウンストリーム システム (REST API、webhook など) の間で要求と応答がどのように流れるかを決定します。
- [P1] デプロイメント: アプリケーションのデプロイ、スケーリング、およびバージョン管理の要件は何ですか?この記事では、Databricks MLflow、Unity Catalog、[Agent Framework](#)、[Agent Evaluation](#)、モデルサービングを使用して、でエンドツーエンドのライフサイクルを処理する方法について説明します。
- [P1] 本番運用 モニタリング & オブザーバビリティ: 本番運用に入ったアプリケーションをどのように監視しますか?
 - ロギング & トレース: 完全な実行トレースをキャプチャします。
 - 品質 メトリクス: ライブトラフィックの主要なメトリクス(正確性、レイテンシ、関連性など)を継続的に評価します。
 - アラート & ダッシュボード: 重大な問題に対するアラートを設定します。
 - フィードバックループ:ユーザーのフィードバックを本番運用に取り入れる(親指を立てる、または下げる)ことで、問題を早期に発見し、反復的な改善を推進します。

例

例として、これらの考慮事項と要件が、 Databricks 顧客サポートチームが使用する架空のエージェント型 RAGアプリケーションにどのように適用されるかを考えてみます。

領域	考慮事項	要件
ユーザーエクスペリエンス	<ul style="list-style-type: none">● インタクションモダリティ。● 一般的なユーザー クエリの例。● 想定される応答の形式とスタイル。● あいまいなクエリや無関係なクエリの処理。	<ul style="list-style-type: none">● Slackと統合されたチャットインターフェース。● クエリの例: "クラスターの起動時間を短縮するにはどうすればよいですか?" "どのようなサポート プランがありますか?"● 明確で技術的な回答と、コードスニペット、および必要に応じて関連ドキュメントへのリンク。● コンテキストに応じた提案を提供し、必要に応じてサポート エンジニアにエスカレーションします。

領域	考慮事項	要件
エージェントロジック	<ul style="list-style-type: none"> クエリの理解と分類。 マルチステップの計画と意思決定。 自律的なツール選択と実行。 インタラクション間の状態とコンテキストの管理。 エラー処理とフォールバックメカニズム。 	<ul style="list-style-type: none"> 決定論的なフォールバックを備えた LLM ベースの計画。 事前定義された一連のツール(ドキュメント取得ツールやSalesforce取得ツールなど)と統合します。 一貫性のあるマルチターンインタラクションと堅牢なエラーリカバリのための会話状態を維持します。
データ	<ul style="list-style-type: none"> データソースの番号と種類。 データ形式と場所。 データ サイズと更新頻度。 データの品質と一貫性。 	<ul style="list-style-type: none"> 4つのデータソース。 会社のドキュメント(HTML、PDF)。 解決されたサポート チケット (JSON)。 コミュニティ フォーラムの投稿 (Delta テーブル)。 Salesforce コネクタ。 データは Unity Catalog に保存され、毎週更新されます。 合計データ サイズ: 5 GB。 専任のドキュメントとサポートチームによって維持される一貫したデータ構造と品質。
パフォーマンス	<ul style="list-style-type: none"> 許容可能な最大レイテンシ。 コストの制約。 予想される使用量と同時実行性。 	<ul style="list-style-type: none"> 最大待機時間の要件。 コストの制約。 予想されるピーク負荷。
評価	<ul style="list-style-type: none"> 評価データセットの可用性。 品質メトリクス。 ユーザーフィードバックの収集。 	<ul style="list-style-type: none"> 各製品領域の対象分野の専門家が、出力のレビューと不正解の調整を支援して評価データセットを作成します。 ビジネス KPI。 サポート チケットの解決率の向上。 サポート チケットあたりのユーザー時間の短縮。 品質メトリクス。

領域	考慮事項	要件
		<ul style="list-style-type: none">• LLMが判断した回答の正しさと関連性。• LLMは検索精度を判定します。• ユーザーが賛成票を投じるか、反対票を投じるか。• フィードバック収集。• Slackは、サムズアップ・サムズダウンを提供するように実装されます。
セキュリティ	<ul style="list-style-type: none">• 機密データの取り扱い。• アクセス制御の要件。	<ul style="list-style-type: none">• 機密性の高い顧客データを取得ソースに含めないでください。• Databricks コミュニティ SSO によるユーザー認証。
デプロイメント	<ul style="list-style-type: none">• 既存のシステムとの統合。• デプロイとバージョン管理。	<ul style="list-style-type: none">• サポートチケットシステムとの統合。• Databricks モデルサービング エンドポイントとしてデプロイされたエージェント。

ソリューション設計

その他の設計上の考慮事項については、 [エージェント・システムの設計パターン](#)を参照してください。

データソースとツール

生成 AI アプリを設計する際には、ソリューションの推進に必要なさまざまなデータソースとツールを特定し、マッピングすることが重要です。これには、構造化データセット、非構造化データ処理パイプライン、または外部 APIのクエリが含まれる場合があります。 以下は、生成AIアプリにさまざまなデータソースやツールを組み込むための推奨されるアプローチです。

構造化データ

構造化データは通常、明確に定義された表形式 (Delta テーブルや CSV ファイルなど) で存在し、クエリが事前に決定されているか、ユーザー入力に基づいて動的に生成する必要があるタスクに最適です。生成AI アプリへの構造化データの追加に関する推奨事項については、 [構造化取得 AI エージェント ツール](#) を参照してください。

非構造化データ

非構造化データには、未加工のドキュメント、PDF、電子メール、画像、および固定スキーマに準拠していないその他の形式が含まれます。このようなデータは、通常、解析、チャンク化、埋め込みの組み合わせによる追加の処理が必要であり、Gen AI アプリで効果的にクエリを実行して使用する必要があります。Gen AI アプリへの構造化データの追加に関する推奨事項については、「[非構造化データの取得ツールの構築とトレース](#)」を参照してください。

外部 API とアクション

一部のシナリオでは、生成 AI アプリは、データを取得したりアクションを実行したりするために、外部システムと対話する必要がある場合があります。アプリケーションでツールの呼び出しや外部 API との対話が必要な場合は、次のことをお勧めします。

- **Unity Catalog Connection を使用して API 資格情報を管理します。** Unity Catalog Connection を使用して、API 資格情報を安全に管理します。この方法により、トークンとシークレットが一元的に管理され、アクセス制御されます。
- **Databricks SDK を使用して呼び出します。**
databricks-sdk ライブラリの `http_request` 関数を使用して、外部サービスに HTTP 要求を送信します。この機能は、認証に Unity Catalog 接続を利用し、標準の HTTP メソッドをサポートします。
- **Unity Catalog の機能を活用します。**
外部接続を Unity Catalog 関数にラップして、カスタムの前処理ロジックまたは後処理ロジックを追加します。
- **Python エグゼキューター ツール：**
API 関数を使用してデータ変換または Python インタラクションのコードを動的に実行するには、組み込み Python エグゼキューター ツールを使用します。

例：

内部アナリティクス アプリケーションは、外部の金融 API からライブ市場データを取得します。このアプリケーションは、次のものを使用します。

- **Unity Catalog External Connection** は、API 資格情報を安全に保存します。
- **カスタム Unity Catalog 関数**は、API 呼び出しをラップして、前処理（データの正規化など）と後処理（エラー処理など）を追加します。
- さらに、アプリケーションは **Databricks SDK を介して API を直接呼び出す**こともできます。

実装アプローチ

生成 AI アプリを開発する場合、エージェントのロジックを実装するための2つの主要なオプションがあります: オープンソースフレームワークを活用するか、Pythonコードを使用してカスタムソリューションを構築するかです。以下は、各アプローチの長所と短所の内訳です。

フレームワーク(LangChain、LlamaIndex、CrewAI、AutoGenなど)を使用する

長所：

- **すぐに使用できるコンポーネント:** フレームワークには、迅速な管理、通話のチェーン化、さまざまなデータソースとの統合のための既製のツールが付属しており、開発を加速できます。
- **コミュニティとドキュメント:** コミュニティのサポート、チュートリアル、定期的な更新を利用できます。
- **一般的な設計パターン:** フレームワークは通常、一般的なタスクを調整するための明確なモジュール構造を提供し、エージェント全体の設計を簡素化できます。

短所：

- **抽象化を追加しました:** オープンソースフレームワークは、多くの場合、特定のユースケースには不要な抽象化レイヤーを導入します。
- **更新への依存性:** バグ修正や機能の更新をフレームワークのメンテナーに依存している可能性があるため、新しい要件に迅速に適応する能力が遅くなる可能性があります。
- **潜在的なオーバーヘッド:** 余分な抽象化は、アプリケーションによりきめ細かな制御が必要な場合に、カスタマイズの課題につながる可能性があります。

純粋なPythonの使用

長所：

- **柔軟性：** 純粋なPythonで開発すると、フレームワークの設計上の決定に制約されることなく、実装をニーズに合わせて正確に調整できます。
- **迅速な適応:** 外部フレームワークからの更新を待たずに、コードをすばやく調整し、必要に応じて変更を組み込むことができます。
- **簡略：** 不要な抽象化レイヤーを避けると、よりスリムでパフォーマンスの高いソリューションになる可能性があります。

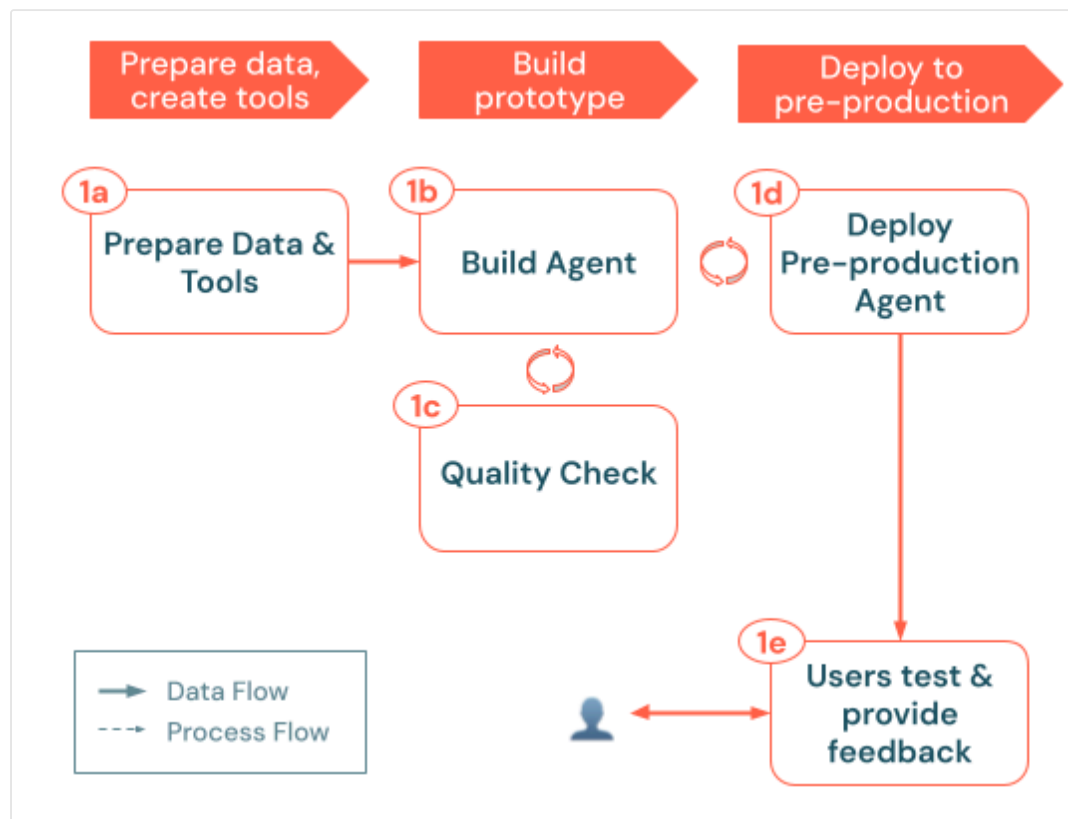
短所：

- **開発作業の増加:** ゼロから構築するには、専用のフレームワークが提供する機能を実装するために、より多くの時間と専門知識が必要になる場合があります。
- **車輪の再発明:** 一般的な機能(ツールチェーンやプロンプト管理など)を自分で開発する必要がある場合があります。
- **メンテナンスの責任:** すべての更新とバグ修正はお客様の責任となりますが、これは複雑なシステムでは困難な場合があります。

最終的には、プロジェクトの複雑さ、パフォーマンスのニーズ、必要な制御のレベルに基づいて決定を下す必要があります。どちらのアプローチも本質的に優れているわけではありません。それぞれに、開発の好みや戦略的な優先順位に応じて、異なる利点があります。

1. ビルド

この段階では、ソリューション設計を現用世代の AI アプリケーションに変換します。すべてを事前に完成させるのではなく、迅速にテストできる最小限の概念実証(POC)から小さく始めます。これにより、本番運用前の環境にできるだけ早くデプロイし、実際のユーザーや専門家から代表的なクエリを収集し、実際のフィードバックに基づいて絞り込むことができます。



ビルド プロセスは、次の主要な手順に従います。

a. **データとツールの準備:** 必要なデータにアクセスでき、解析され、取得できる状態になっていることを確認します。エージェントが必要とする Unity Catalog 機能と接続 (取得 API や外部 API コールなど) を実装または登録するb. **ビルド エージェント:** シンプルなPOCアプローチから始めて、コアロジックを調整します。c. **品質チェック:** アプリをより多くのユーザーに公開する前に、重要な機能を検証します。d. **本番運用前のエージェントをデプロイします。** POC をテスト ユーザーと対象分野の専門家が初期フィードバックに利用できるようにします。e. **ユーザーからのフィードバックを収集する:** 実際の使用状況を使用して、改善領域、必要な追加データまたはツール、および次のイテレーションのための潜在的な改良点を特定します。

ある。データとツールの準備

ソリューションの設計フェーズから、アプリに必要なデータソースとツールの最初のアイデアが得られます。この段階では、POCを検証するのに十分なデータだけに焦点を当て、最小限に抑えます。これにより、複雑なパイプラインに多額の先行投資をすることなく、迅速なイテレーションが可能になります。

データ

1. データの代表的なサブセットを特定する

- **構造化データ** の場合は、初期シナリオに最も関連性の高い主要なテーブルまたは列を選択します。
- **非構造化データ** の場合は、代表的なドキュメントのサブセットのみをインデックス化することを優先します。[Mosaic AI Vector Search](#)で基本的なチャンク/埋め込みパイプラインを使用すると、エージェントは必要に応じて関連するテキストチャンクを取得できます。

2. データアクセスを設定する

- アプリで外部 API 呼び出しを行う必要がある場合は、[Unity Catalog 接続](#)を使用して資格情報を安全に保存してください。

3. 基本的なカバレッジの検証

- 選択したデータのサブセットが、テストする予定のユーザークエリに適切に対応していることを確認します。
- 追加のデータソースや複雑な変換は、将来のイテレーションのために保存してください。現在の目標は、基本的な実現可能性を証明し、フィードバックを収集することです。

ツール

データソースを設定したら、次のステップは、エージェントがランタイムで呼び出すツールを実装して登録することです。Unity Catalog ツールは、SQL クエリや外部 API 呼び出しなどの **1つの対話機能** であり、エージェントは取得、変換、またはアクションのために呼び出すことができます。

データ取得ツール

- **制約のある構造化データクエリ:** クエリが固定されている場合は、[Unity Catalog SQL 関数](#)または [Python UDF](#) でラップします。これにより、ロジックが一元化され、検出可能に保たれます。
- **自由形式の構造化データクエリ:** クエリがよりオープンエンドの場合は、テキストから SQL へのクエリを処理するための [Genieスペース](#) を設定することを検討してください。
- **非構造化データヘルパー関数:** Mosaic AI Vector Searchに保存されている非構造化データの場合は、エージェントが関連するテキストチャンクを取得するために呼び出すことができる [非構造化データ取得ツールを作成します](#)。

API呼び出しツール

- **外部 API 呼び出し:** [API 呼び出し](#)は、Databricks SDK の [http_request](#) メソッドを使用して直接呼び出すことができます。
- **オプションのラッパー:** 前処理または後処理ロジック (データの正規化やエラー処理など) を実装する必要がある場合は、API 呼び出しを [Unity Catalog 関数](#)にラップします。

最小限に抑える

- **基本的なツールのみから始めます。** 1つの取得パスまたは限られた API 呼び出しのセットに焦点を当てます。イテレーションのたびに、さらに追加できます。
- **対話形式で検証します。** 各ツールをエージェント システムに組み込む前に、個別に (ノートブックなどで) テストします。

プロトタイプツールの準備ができたら、エージェントのビルドに進みます。エージェントは、これらのツールを調整して、クエリに応答し、データをフェッチし、必要に応じてアクションを実行します。

b.ビルド エージェント

データとツールを配置したら、ユーザークエリなどの受信リクエストに応答するエージェントを構築できます。最初のプロトタイプ エージェントを作成するには、[Python](#) または [AI プレイグラウンド](#)を使用します。以下の手順に従います。

1. シンプルに始める

- **基本的なデザインパターンを選択します。** POC の場合は、基本的なチェーン (固定された一連のステップなど) または1つのツール呼び出しエージェント (LLM が1つまたは 2 つの重要なツールを動的に呼び出すことができる) から開始します。
 - シナリオが Databricksドキュメントで提供されているサンプル ノートブックのいずれかと一致する場合は、そのコードをスケルトンとして適応させます。
- **最小限のプロンプト:** この時点で、プロンプトを過剰に設計する衝動を抑えてください。指示は簡潔で、最初のタスクに直接関連するものにしてください。

2. ツールを組み込む

- **ツールの統合:** チェーン デザイン パターンを使用する場合、各ツールを呼び出すステップはハードコーディングされます。ツール呼び出しエージェントでは、LLM が関数を呼び出す方法を認識できるように [スキーマを指定します](#)。
 - ツールを単独で想定どおりに動作していることを検証してから、エージェントシステムに組み込み、エンドツーエンドでテストします。
- **ガードレール:** エージェントが外部システムを変更したり、コードを実行したりできる場合は、基本的な安全性チェックとガードレール (呼び出し回数の制限や特定のアクションの制限など) があることを確認してください。これらを [Unity Catalog 関数](#)内に実装します。

3. MLflow を使用してエージェントをトレースしてログに記録する

- **各ステップをトレースします。** [MLflow トレース](#)を使用して、ステップごとの入力、出力、経過時間をキャプチャして、問題をデバッグし、パフォーマンスを測定します。

- 。エージェントをログに記録します。 [MLflow Tracking](#) を使用して、エージェントのコードと構成をログに記録します。

この段階では、**完璧さが目標ではありません**。テストユーザーや SME からの早期フィードバックのためにデプロイできる、シンプルで機能するエージェントが必要です。次のステップは、本番運用前の環境で利用可能にする前に、迅速な品質チェックを実行することです。

c. 品質チェック

エージェントを本番運用前の幅広いオーディエンスに公開する前に、エンドポイントにデプロイする前に、オフラインで「十分」な品質チェックを実行して、主要な問題を発見してください。この段階では、通常、大規模で堅牢な評価データセットはありませんが、エージェントがいくつかのサンプル クエリで意図したとおりに動作することを確認するには、クイック パスを実行できます。

1. ノートブックでの対話型のテスト

- 。 **手動検査:** 担当者に手動で電話して、担当者のリクエストを伝えます。正しいデータを取得し、ツールを正しく呼び出し、目的の形式に従っているかどうかに注意してください。
- 。 **MLflow トレースを検査します。** MLflow トレースを有効にしている場合は、ステップバイステップのテレメトリを確認します。エージェントが適切なツールを選択し、エラーを適切に処理し、予期しない中間要求や結果を生成しないことを確認します。
- 。 **遅延を確認します。** 各要求の実行にかかる時間に注意してください。応答時間やトークンの使用量が長すぎる場合は、先に進む前にステップを整理したり、ロジックを簡略化したりする必要があります。

2. バイブチェック

- 。これは、ノートブックまたは [AI Playground](#) で実行できます。
- 。 **一貫性と正確性:** エージェントの出力は、テストしたクエリに対して意味がありますか? 明らかな不正確さや欠落している詳細はありませんか?
- 。 **エッジケース:** 人里離れたクエリをいくつか試した場合でも、エージェントは論理的に応答しましたか、それとも少なくとも優雅に失敗しましたか (たとえば、無意味な出力を生成するのではなく、丁寧に回答を拒否しました)?
- 。 **迅速な遵守:** 希望のトーンやフォーマットなどの高レベルの指示を提供した場合、エージェントはこれらに従っていますか?

3. 「十分」の品質を評価する

- 。この時点でテストクエリが限られている場合は、合成データの生成を検討してください。 [「評価セットを作成する」](#) を参照してください。
- 。 **主要な問題に対処します。** 重大な欠陥 (たとえば、エージェントが無効なツールを繰り返し呼び出したり、無意味な出力を出力したりするなど) が見つかった場合は、これらの問題を修正してか

ら、より多くのユーザーに公開してください。「[一般的な品質の問題とその修正方法](#)」を参照してください。

- **実行可能性を決定します。** エージェントが小さなクエリセットの使いやすさと正確性の基本的な基準を満たしている場合は、続行できます。そうでない場合は、プロンプトを絞り込み、ツールまたはデータの問題を修正して、再テストします。

4. 次のステップを計画する

- **トラックの改善:** 延期することにした欠点は文書化します。本番運用前に実社会のフィードバックを集めたら、それらを再検討することができます。

すべてが限定的なロールアウトで実行可能に見える場合は、エージェントを本番運用前にデプロイする準備ができています。徹底的な評価プロセスは、特に実際のデータ、SMEのフィードバック、構造化された評価セットが得られた後、[後のフェーズ](#)で行われます。今のところは、エージェントがそのコア機能を実際に発揮できるようにすることに集中してください。

d. 本番運用エージェントのデプロイ

エージェントがベースライン品質のしきい値を満たしたら、次のステップは、ユーザーがアプリをどのようにクエリし、開発をガイドするためのフィードバックを収集するかを理解できるように、本番運用前の環境でエージェントをホストすることです。この環境は、POC フェーズ中の開発環境にすることができます。主な要件は、内部テスターまたはドメイン専門家が選択できる環境にアクセスできることです。

1. エージェントをデプロイする

- **エージェントの記録と登録:** まず、MLflowモデルとして[エージェント](#)を記録し、Unity Catalogに[登録](#)します。
- **Agent Framework を使用してデプロイします。** Agent Framework を使用して、登録済みのエージェントを取得し、モデルサービングエンドポイントとして [デプロイ](#) します。

2. 推論テーブル

- Agent Framework は、リクエスト、レスポンス、トレースをメタデータとともに、各サービングエンドポイントの Unity Catalog の [推論テーブル](#) に自動的に保存します。

3. セキュリティ保護と構成

- **アクセス制御:** [エンドポイントへのアクセス](#) をテストグループ (SME、パワーユーザー) に制限します。これにより、制御された使用が保証され、予期しないデータ漏洩が回避されます。
- **認証:** 必要なシークレット、API トークン、またはデータベース接続が適切に構成されていることを確認します。

これで、実際のクエリに関するフィードバックを収集するための制御された環境ができました。エージェントとすばやくやり取りする方法の1つは、[AI Playground](#)で、新しく作成したモデルサービングエンドポイントを選択し、エージェントにクエリを実行することです。

e.ユーザーからのフィードバックを収集する

エージェントを本番運用前の環境にデプロイしたら、次のステップは、実際のユーザーや SME からフィードバックを収集して、ギャップを明らかにし、不正確さを特定し、エージェントをさらに洗練させることです。

1. レビューアプリを使用する

- Agent Frameworkを使用してエージェントをデプロイすると、シンプルなチャットスタイルの [レビューアプリ](#) が作成されます。テスターが質問を投げかけ、エージェントの回答をすぐに評価できるユーザーフレンドリーなインターフェースを提供します。
- すべてのリクエスト、レスポンス、ユーザーフィードバック(親指を立てたり下げたり、コメントを書いたり)は、 [自動的に推論テーブル](#) に記録されるため、後で簡単に分析できます。

2. モニタリング UI を使用してログを検査する

- [モニタリング UI](#) で賛成票/反対票またはテキストフィードバックを追跡して、テスターが特に役に立った (または役に立たなかった) 回答を確認します。

3. ドメインエキスパートの関与

- 専門家に典型的なシナリオと珍しいシナリオを実行するように促します。ドメイン知識は、ポリシーの誤解やデータの欠落などの微妙なエラーを表面化するのに役立ちます。
- プロンプトによる小さな調整から大規模なデータパイプラインのリファクタリングまで、問題のバックログを保持します。次に進む前に、どの修正を優先するかを決定します。

4. 新しい評価データのキュレーション

- 注目すべきインタラクションや問題のあるインタラクションをテストケースに変換する。時間が経つにつれて、これらはより堅牢な評価データセットの基礎を形成します。
- 可能であれば、これらのケースに正しい答えまたは予想される答えを追加します。これは、後続の評価サイクルで品質を測定するのに役立ちます。

5. フィードバックに基づいて反復する

- 小さなプロンプトの変更や新しいガードレールなどの迅速な修正を適用して、差し迫った問題点に対処します。
- 高度なマルチステップ ロジックや新しいデータソースなど、より複雑な問題については、大規模なアーキテクチャ変更に投資する前に十分な証拠を収集します。

この本番運用前のフェーズでは、Review App、推論テーブルログ、SMEの知見からのフィードバックを活用することで、主要なギャップを表面化し、エージェントを反復的に改善することができます。このステップで収集された実際の相互作用は、構造化された評価セットを構築するための基盤を作成し、アドホックな改善から品質測定へのより体系的なアプローチへの移行を可能にします。繰り返し発生する問題に対

処し、パフォーマンスが安定したら、しっかりとした評価が実施された本番運用のデプロイに十分な準備が整います。

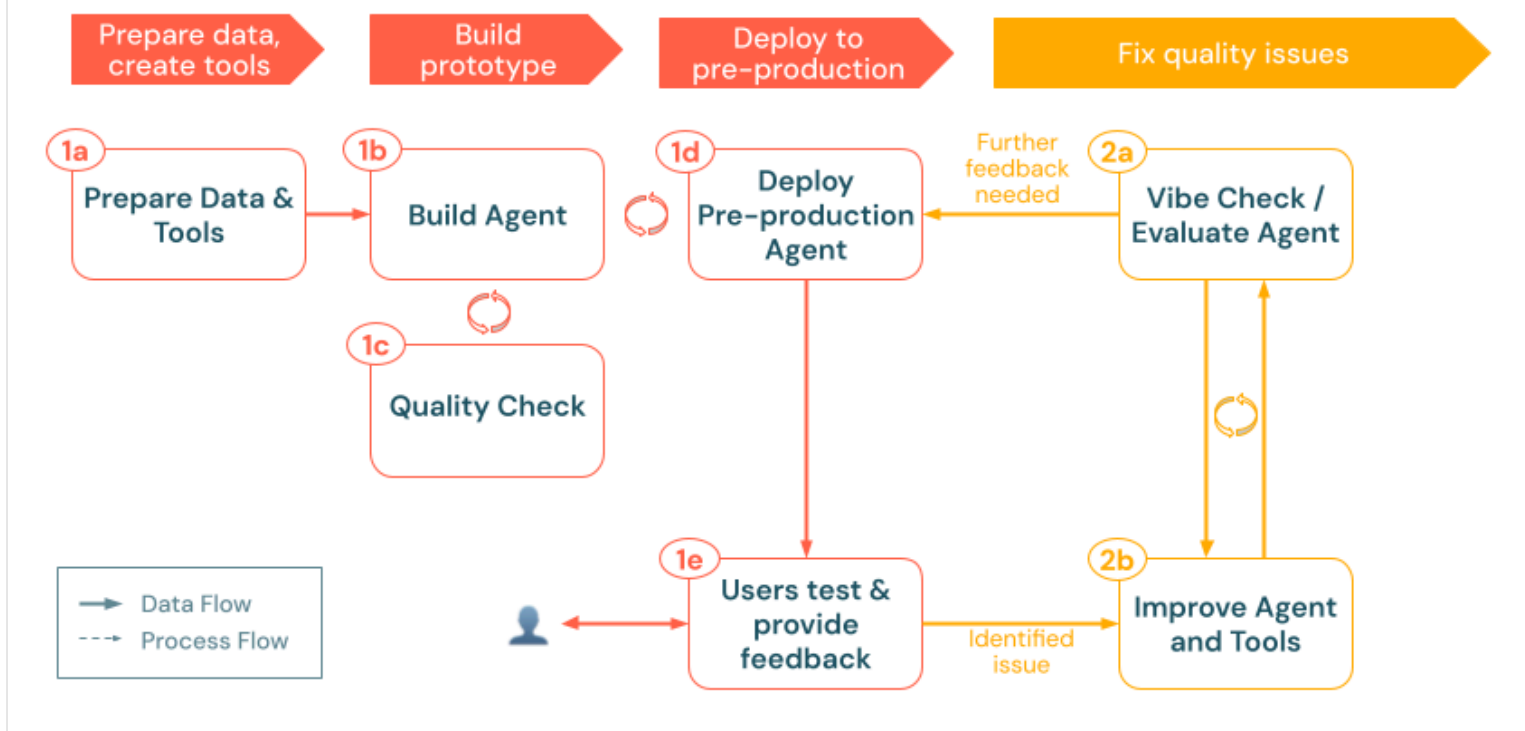
2. 評価と反復

生成AIアプリを本番運用前の環境でテストしたら、次のステップは、その品質を体系的に測定、診断、改善することです。この「評価と反復」フェーズでは、未加工のフィードバックとログが構造化された評価セットに変換されるため、改善を繰り返しテストし、アプリが精度、関連性、安全性の必要な基準を満たしていることを確認できます。

このフェーズには、次の手順が含まれます。

- **ログから実際のクエリを収集します。** 価値の高いインタラクションや問題のあるインタラクションを推論テーブルからテストケースに変換します。
- **エキスパートラベルを追加します。** 可能であれば、これらのケースにグラウンドトゥールズまたはスタイルとポリシーのガイドラインを添付して、正確性、根拠性、およびその他の品質側面をより客観的に測定できるようにします。
- **レバレッジエージェントの評価:** 組み込みの LLM ジャッジまたはカスタム チェックを使用して、アプリの品質を定量化します。
- **反復 処理:** エージェントのロジック、データパイプライン、またはプロンプトを洗練させることで、品質を向上させます。評価を再実行して、主要な問題が解決したかどうかを確認します。

これらの機能は、生成 AI アプリが **Databricks の外部で実行されている場合でも機能する** ことに注意してください。MLflow Tracingを使用してコードを計測可能にすることで、あらゆる環境からトレースをキャプチャし、それらをDatabricks Data Intelligence Platformに統合して、一貫した評価とモニタリングを行うことができます。新しいクエリ、フィードバック、SME の知見を取り入れ続けると、評価データセットは継続的な改善サイクルを支える生きたリソースとなり、生成 AI アプリの堅牢性、信頼性、ビジネス目標への整合性が維持されます。



ある。エージェントの評価

エージェントが本番運用前の環境で稼働した後、次のステップは、アドホックな雰囲気チェックを超えて、そのパフォーマンスを体系的に測定することです。 [Mosaic AI Agent Evaluation](#) は、評価セットの作成、組み込みまたはカスタムの LLM ジャッジによる品質チェックの実行、問題領域の迅速な反復処理を可能にします。

オフラインおよびオンライン評価

生成AIアプリケーションを評価する際には、オフライン評価とオンライン評価の2つの主要なアプローチがあります。開発サイクルのこのフェーズでは、オフライン評価、つまりライブ ユーザーとの対話以外の体系的な評価に焦点を当てています。オンライン評価については、後で本番運用でエージェントのモニタリングについて説明する際に説明します。

チームは、開発者のワークフローで「バイブテスト」に長時間依存しすぎる 경우가多く、非公式にいくつかのクエリを試し、応答が妥当かどうかを主観的に判断します。これは出発点となりますが、本番運用品質のアプリケーションを構築するために必要な厳密さとカバレッジが不足しています。

対照的に、適切なオフライン評価プロセスでは、次のことが行われます。

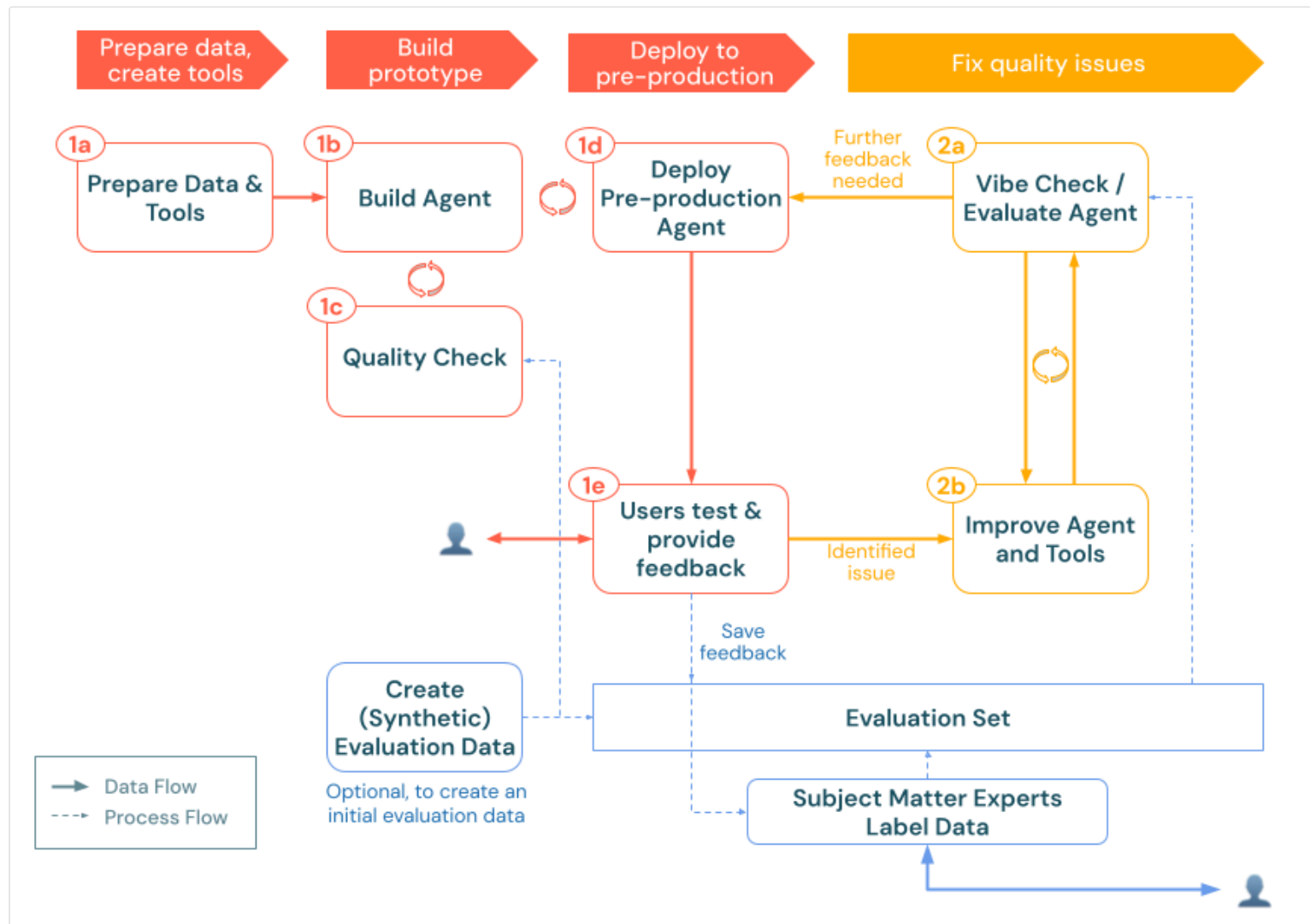
- より広範な展開の前に **品質ベースラインを確立し**、改善の対象となる明確なメトリクスを作成します。
- 注意が必要な **特定の弱点を特定し**、予想されるユースケースのみをテストするという制限を超えます。
- バージョン間でパフォーマンスを自動的に比較することで、アプリを改良する際の **品質の低下を検出** します。
- ステークホルダーに改善を示すための **定量的なメトリクスを提供** します。

- **エッジケース** と潜在的な障害モードをユーザーが気付く前に発見するのに役立ちます。
- パフォーマンスの低いエージェントを本番運用にデプロイする **リスクを軽減**します。

オフライン評価に時間を投資すると、長期にわたる実行に大きな利益がもたらされ、一貫して高品質の回答を提供するのに役立ちます。

評価セットの作成

評価セットは、生成 AI アプリのパフォーマンスを測定するための基盤として機能します。従来のソフトウェア開発のテストスイートと同様に、この代表的なクエリと予想される応答のコレクションは、品質ベンチマークと回帰テストのデータセットになります。



評価セットは、いくつかの補完的なアプローチで構築できます。

1. 推論テーブル ログを評価例に変換する

最も価値のある評価データは、実際の使用状況から直接得られます。本番運用前のデプロイで、リクエスト、エージェントのレスポンス、ツールコール、および取得したコンテキストを含む推論テーブルログが生成されました。

これらのログを評価セットに変換すると、いくつかの利点があります。

- **実際のカバレッジ:** 予想していなかった可能性のある予測不可能なユーザー行動も含まれます。
- **問題に焦点を当てる:** 否定的なフィードバックや応答の遅さに特化したフィルタリングを行うことができます。
- **代表的な分布:** さまざまなクエリタイプの実際の頻度がキャプチャされます。

2. 合成評価データの生成

キュレーションされたユーザークエリのセットがない場合は、[合成評価データセットを自動生成](#)できます。このクエリの「スターターセット」は、エージェントが次のことを行っているかどうかを迅速に評価するのに役立ちます。

- 一貫性のある正確な回答を返します。
- 適切な形式で応答します。
- 構造、調性、およびポリシーガイドラインを尊重します。
- コンテキストを正しく取得します (RAG の場合)。

通常、合成データは完璧ではありません。一時的な足がかりと考えてください。また、次のことも行う必要があります。

- SMEまたはドメインの専門家にレビューしてもらい、無関係なクエリや繰り返しのクエリを整理します。
- 後で実際の使用ログに置き換えたり、拡張したりします。

3. クエリを手動でキュレーションする

合成データに依存しない場合、または推論ログがまだない場合は、10 から 15 の実際のクエリまたは代表的なクエリを特定し、これらから評価セットを作成します。代表的なクエリは、ユーザーインタビューや開発者のブレインストーミングから得られる場合があります。短くて厳選されたリストでさえ、エージェントの応答の明らかな欠陥を明らかにする可能性があります。

これらのアプローチは相互に排他的ではなく、補完的なものです。 効果的な評価セットは時間とともに進化する、通常は次のような複数のソースからの例を組み合わせています。

- 手動でキュレーションされた例から始めて、コア機能をテストします。
- 必要に応じて、合成データを追加して、実際のユーザーデータを取得する前にカバレッジを広げます。
- 実際のログが利用可能になったら、徐々に組み込みます。
- 変化する使用パターンを反映した新しい例で継続的に更新します。

評価クエリのベスト プラクティス

評価セットを作成するときは、次のようなさまざまなクエリの種類を意図的に含めます。

- 予想される使用パターンと予想しない使用パターン (非常に長い要求や短い要求など) の両方。

- 誤用の試みやプロンプト インジェクション攻撃 (システム プロンプトを表示しようとする試みなど) の可能性。
- 複数の推論手順またはツール呼び出しを必要とする複雑なクエリ。
- 最小限の情報やあいまいな情報 (スペルミスやあいまいなクエリなど) を含むエッジケース。
- さまざまなユーザーのスキルレベルと背景を表す例。
- 回答の潜在的なバイアスをテストするクエリ (「会社 A と会社 B を比較する」 など)。

評価セットは、アプリケーションと共に成長し、進化する必要があることを忘れないでください。新しい障害モードやユーザーの行動が明らかになったら、代表的な例を追加して、エージェントがそれらの領域を継続的に改善できるようにします。

評価基準の追加

各評価例には、品質を評価するための基準が必要です。これらの基準は、エージェントの回答を測定する基準として機能し、複数の品質次元にわたる客観的な評価を可能にします。

グラウンドトゥールースファクトまたは参照回答

事実の正確性を評価する際には、主に2つのアプローチがあります:予想される事実と参照回答です。それぞれが評価戦略において異なる目的を果たします。

想定される事実を使用する (推奨)

[expected_facts](#)アプローチでは、正しい回答に表示すべき重要な事実をリストアップします。例については、[request](#)、[response](#)、[guidelines](#)、[expected_facts](#)を含む[サンプル評価セット](#)を参照してください。

このアプローチには、次のような大きな利点があります。

- 応答で事実を表現する方法に柔軟性を持たせることができます。
- 専門家がグラウンドトゥールースを提供しやすくなります。
- さまざまな応答スタイルに対応しながら、コア情報が存在することを確認します。
- モデルのバージョンやパラメーター設定間でより信頼性の高い評価を可能にします。

組み込みの正確性ジャッジは、言い回し、順序付け、または追加のコンテンツに関係なく、エージェントの応答にこれらの重要な事実が組み込まれているかどうかを確認します。

予期される応答を使用する (代替)

または、[完全な参照回答を提供する](#)こともできます。このアプローチは、次の状況で最適に機能します。

- 専門家が作成したゴールドスタンダードの答えがあります。
- 回答の正確な文言や構造が重要です。

- 規制の厳しい状況で回答を評価しています。

Databricks では、精度を確保しながら柔軟性を高めるため、一般的に `expected_response` よりも `expected_facts` を使用することをお勧めします。

スタイル、トーン、またはポリシーのコンプライアンスに関するガイドライン

事実の正確性だけでなく、回答が特定のスタイル、トーン、またはポリシー要件に準拠しているかどうかを評価する必要がある場合があります。

ガイドラインのみ

事実の正確性ではなく、スタイルやポリシー要件を強制することが主な関心事である場合は、[想定される事実を示さずにガイドラインを提供](#)できます。

Python

```
# Per-query guidelines
eval_row = {
    "request": "How do I delete my account?",
    "guidelines": {
        "tone": ["The response must be supportive and non-judgmental"],
        "structure": ["Present steps chronologically", "Use numbered lists"]
    }
}

# Global guidelines (applied to all examples)
evaluator_config = {
    "databricks-agent": {
        "global_guidelines": {
            "rudeness": ["The response must not be rude."],
            "no_pii": ["The response must not include any PII information (personally identifiable information)."]
        }
    }
}
```

ガイドラインLLMの審査員は、これらの自然言語の指示を解釈し、応答がそれらに準拠しているかどうかを評価します。これは、トーン、フォーマット、組織のポリシーの遵守などの主観的な品質次元に特に効果的です。

グラウンドトゥールースとガイドラインの組み合わせ

事前にキャプチャされた応答を使用する

- エージェントの行動の既存のパターンを分析します。
- 以前のバージョンに対するパフォーマンスのベンチマーク。
- 応答を再生成しないことで時間とコストを節約します。
- Databricks の外部で提供されるエージェントの評価。

評価基準のベストプラクティス

2. **ユーザー価値に焦点を当てる:** ユーザーにとって最も重要なことに沿った条件を優先します。

3. **簡単に始めましょう:** 主要な基準セットから始めて、品質ニーズに対する理解が深まるにつれて拡大します。
4. **バランスカバレッジ:** 品質のさまざまな側面(事実の正確性、スタイル、安全性など)に対応する基準を含めます。
5. **フィードバックに基づいて反復します。** ユーザーからのフィードバックと進化する要件に基づいて基準を絞り込みます。

高品質の評価データセットの構築の詳細については、「[評価セットを開発するためのベスト プラクティス](#)」を参照してください。

評価の実行

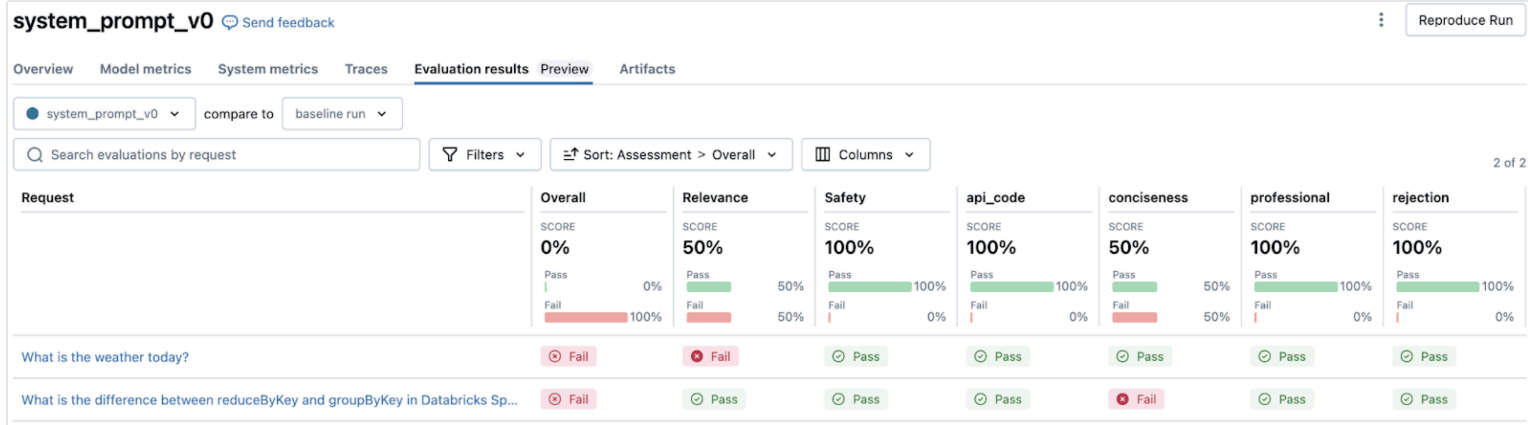
クエリと条件を含む評価セットを準備したので、`mlflow.evaluate()` を使用して評価を実行できます。この機能は、エージェントの呼び出しから結果の分析まで、評価プロセス全体を処理します。

基本的な評価ワークフロー

基本的な評価を実行するのに必要なのは、わずか数行のコードです。詳細については、「[評価を実行する](#)」を参照してください。

評価がトリガーされると、次のようになります。

1. 評価セットの各行について、`mlflow.evaluate()` は次の処理を行います。
 - クエリを使用してエージェントに電話をかけます(まだ応答を提供していない場合)。
 - 組み込みの LLM ジャッジを適用して、品質ディメンションを評価します。
 - トークンの使用量やレイテンシなどの運用メトリクスを計算します。
 - 各評価の詳細な根拠を記録します。
2. 結果は MLflow に自動的に記録され、次のものが作成されます。
 - 行ごとの品質評価。
 - すべての例で集計されたメトリクス。
 - デバッグと分析のための詳細なログ。



評価のカスタマイズ

追加のパラメーターを使用して、特定のニーズに合わせて評価を調整できます。 `evaluator_config` パラメーターを使用すると、次の操作を実行できます。

- 実行する組み込みジャッジを選択します。
- すべての例に適用されるグローバル ガイドラインを設定します。
- ジャッジのしきい値を設定します。
- ジャッジの評価を導くために、いくつかのショットの例を提供します。

詳細と例については、「[例](#)」を参照してください。

Databricks の外部でエージェントを評価する

Agent Evaluation の強力な機能の 1 つは、Databricks だけでなく、どこにでもデプロイされた 生成AIアプリを評価できることです。

どの裁判官が適用されるか

デフォルトでは、Agent Evaluation は、評価セットで利用可能なデータに基づいて、[適切な LLM ジャッジを自動的に選択します](#)。品質の評価方法の詳細については、「[LLM 審査員による品質の評価方法](#)」を参照してください。

評価結果の分析

評価を実行すると、MLflow UI でアプリのパフォーマンスを理解するための視覚化と分析が提供されます。この分析は、パターンの特定、問題の診断、改善の優先順位付けに役立ちます。

評価結果のナビゲート

`mlflow.evaluate()`, を実行した後に MLflow UI を開くと、相互接続されたビューがいくつか見つかります。MLflow UI でこれらの結果をナビゲートする方法については、「[MLflow UI を使用して出力を確認する](#)」を参照してください。

障害パターンを解釈する方法のガイダンスについては、「[b. エージェントとツールを改善する](#)」を参照してください。

カスタムAIのジャッジとメトリクス

組み込みのジャッジは、多くの一般的なチェック(正確性、スタイル、ポリシー、安全性など)を対象としていますが、アプリのパフォーマンスのドメイン固有の側面を評価する必要がある場合があります。カスタムジャッジとメトリクスにより、評価機能を拡張して、独自の品質要件に対応できます。

カスタムLLMジャッジ

プロンプトからカスタム LLM ジャッジを作成する方法の詳細については、「[プロンプトから AI ジャッジを作成する](#)」を参照してください。

カスタムジャッジは、次のような人間のような判断から恩恵を受ける主観的または微妙な品質次元の評価に優れています。

- ドメイン固有のコンプライアンス(法律、医療、財務)。
- ブランドの声とコミュニケーションスタイル。
- 文化的な感受性と適切さ。
- 複雑な推論の品質。
- 専門的なライティング規約。

ジャッジの出力は、組み込みのジャッジと共に MLflow UI に表示され、評価を説明するのと同じ詳細な根拠が表示されます。

カスタムメトリクス

よりプログラムの決定論的な評価を行うには、`@metric` デコレーターを使用してカスタムメトリクスを作成できます。「[@metric デコレータ](#)」を参照してください。

カスタムメトリクスは、次の場合に最適です。

- 形式の検証やスキーマのコンプライアンスなどの技術要件の検証。
- 特定のコンテンツの有無を確認する。
- 応答の長さや複雑さのスコアなどの定量的測定を実行します。
- ビジネス固有の検証ルールを実装する。
- 外部検証システムとの統合。

b. エージェントとツールの改善

評価を行い、品質問題を特定したら、次のステップは、それらの問題に体系的に対処してパフォーマンスを向上させることです。評価結果から、エージェントがどこでどのように障害が発生しているかについての

貴重な知見が得られ、ランダムな調整ではなく、的を絞った改善を行うことができます。

一般的な品質問題とその修正方法

評価結果からLLMジャッジの評価は、エージェントシステムの特定のタイプの障害を指し示しています。このセクションでは、これらの一般的な障害パターンとその解決策について説明します。LLMジャッジ出力の解釈方法については、「[ジャッジ出力AI](#)」を参照してください。

品質イテレーションのベスト プラクティス

改善を反復しながら、厳密なドキュメントを維持します。例えば：

1. 変更のバージョン管理

- MLflow Tracking を使用して、重要な各イテレーションをログに記録します。
- プロンプト、構成、およびキー・パラメーターを中央の構成ファイルに保存します。これがエージェントに記録されていることを確認します。
- 新しくデプロイされたエージェントごとに、変更内容とその理由を詳しく説明する [変更ログ](#) をリポジトリに保持します。

2. うまくいったこととうまくいかなかったことの両方を文書化する

- 成功したアプローチと失敗したアプローチの両方を文書化します。
- 各変更がメトリクスに与える具体的な影響に注意してください。エージェント評価 MLflow の実行にリンクし直します。

3. ステークホルダーと連携する

- レビューアプリを使用して、SMEの改善を検証します。
 - [レビュー担当者の指示](#)を使用して、変更をレビュー担当者に伝えます。
- エージェントの異なるバージョンを並べて比較するには、複数のエージェント エンドポイントを作成し、[AI Playground](#) でモデルを使用することを検討してください。これにより、ユーザーは同じリクエストを別々のエンドポイントに送信し、レスポンスとトレースを並べて調べることができます。

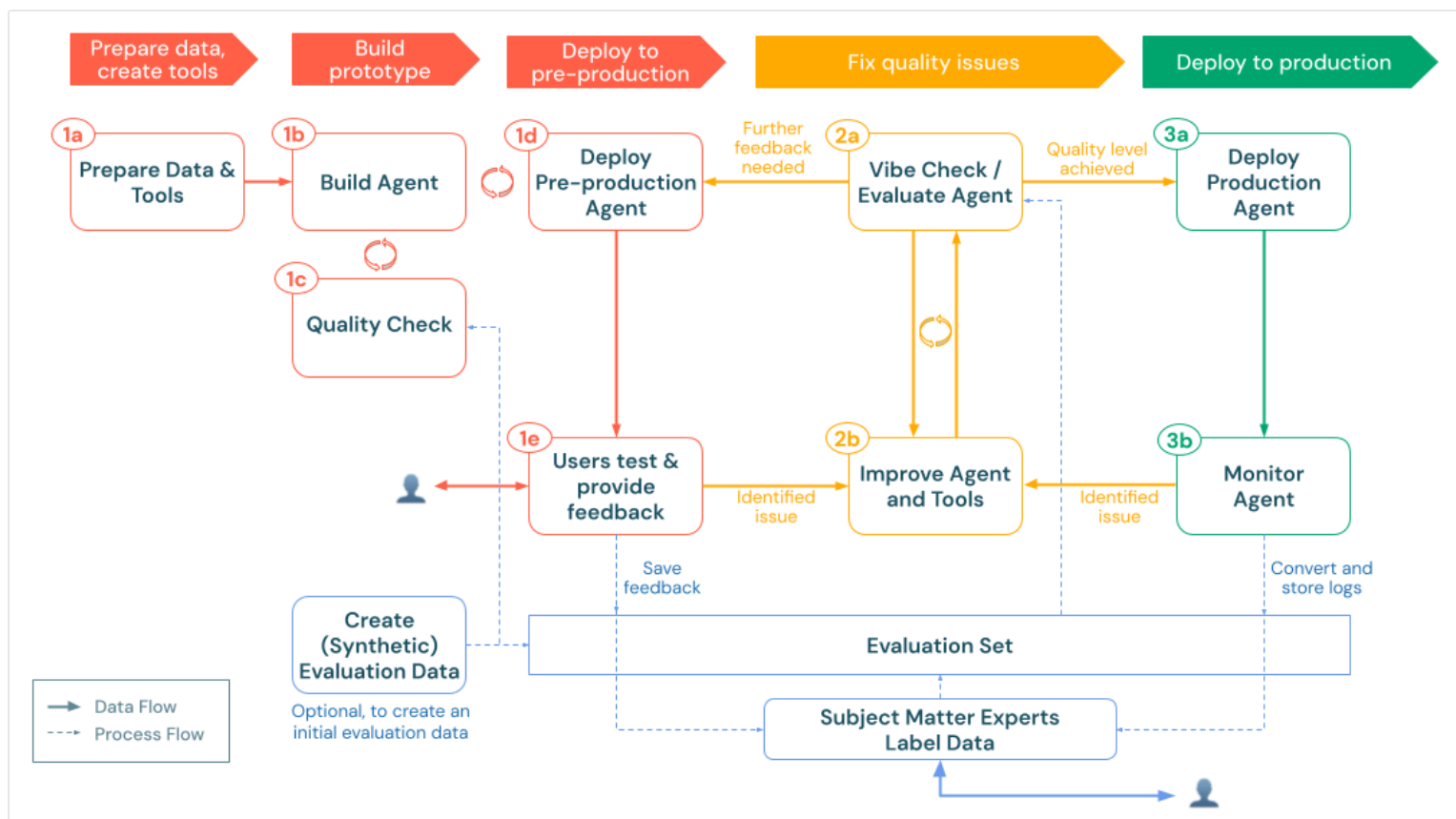
3. 本番運用

アプリを繰り返し評価して改善した結果、要件を満たす品質レベルに到達し、より広範な使用が可能になりました。本番運用フェーズでは、改良したエージェントを本番運用環境にデプロイし、継続的なモニタリングを実施して、品質を長期にわたって維持します。

本番運用フェーズには、次のものが含まれます。

- **Deploy agent to 本番運用:** 適切なセキュリティ、スケーリング、および認証設定を備えた本番運用可能なエンドポイントを設定します。
- **本番運用におけるエージェントの監視:** 継続的な品質評価、パフォーマンス追跡、およびアラートを確立して、エージェントが実際の使用で高い品質と信頼性を維持できるようにします。

これにより、モニタリングの知見がさらなる改善を促進する継続的なフィードバックループが生まれ、それをテスト、デプロイ、および監視し続けることができます。このアプローチにより、アプリは高品質でコンプライアンスを維持し、ライフサイクル全体を通じて進化するビジネス ニーズに合わせることができます。



ある。Deploy agent to 本番運用

徹底的な評価と反復的な改善が完了したら、エージェントを本番運用環境にデプロイする準備が整います。[Mosaic AI エージェントフレームワーク](/generative-ai/agent-framework/build-gen AI-apps.md#agent-framework) 多くのデプロイの問題を自動的に処理することで、このプロセスを簡素化します。

デプロイプロセス

エージェントを本番運用にデプロイするには、次の手順が必要です。

1. エージェントを MLflowModel として に 記録して登録するUnity Catalog
2. Agent Framework を使用してエージェントをデプロイします。
3. エージェントがアクセスする必要がある依存リソースの認証を構成します。
4. デプロイメントをテストして、本番運用環境で機能を確認します。

- 。モデルサービング エンドポイントの準備ができれば、AI Playgroundでエージェントと対話し、機能をテストおよび検証できます。

実装手順の詳細については、「[生成AI アプリケーション用のエージェントをデプロイする](#)」を参照してください。

本番運用デプロイメントに関する考慮事項

本番運用に移行する際には、以下の重要な考慮事項に留意してください。

パフォーマンスとスケーリング

- 。予想される使用パターンに基づいて、コストとパフォーマンスのバランスを取ります。
- 。断続的に使用されるエージェントに対して [ゼロへのスケール](#) を有効にすることを検討して、コストを削減することを検討してください。
- 。アプリケーションのユーザーエクスペリエンスのニーズに基づいてレイテンシー要件を理解します。

セキュリティとガバナンス

- 。すべてのエージェント コンポーネントに対して、Unity Catalog レベルで適切なアクセス制御を確保します。
- 。可能な場合は、Databricks リソースに [組み込みの認証パススルー](#) を使用します。
- 。外部 API またはデータソースの適切な資格情報管理を構成します。

統合アプローチ

- 。アプリケーションがエージェントとどのように対話するかを決定します (たとえば、API や組み込みインターフェイスを使用)。
- 。アプリケーションでエージェントの応答を処理して表示する方法を検討します。
 - 。クライアントアプリケーションに追加のコンテキスト (ソースドキュメント参照や信頼度スコアなど) が必要な場合は、このメタデータをレスポンスに含めるようにエージェントを設計します ([カスタム出力](#)を使用するなど)。
- 。エラー処理と、エージェントが使用できない場合のフォールバックメカニズムを計画します。

フィードバック収集

- 。レビューアプリを活用して、初期ロールアウト中に関係者のフィードバックを得ます。
- 。ユーザーフィードバックをアプリケーションインターフェイスで直接収集するメカニズムを設計します。

- 。レビューアプリからフィードバックを収集するために作成されたフィードバックエンドポイントは、外部アプリケーションがエージェントに関するフィードバックを提供するためにも使用できます。「[デプロイされたエージェントに関するフィードバックを提供する](#)」を参照してください。
- 。フィードバックデータが評価および改善プロセスに流れるようにします。

b. 本番運用におけるエージェントの監視

エージェントが本番運用にデプロイされた後、そのパフォーマンス、品質、および使用パターンを継続的に監視することが不可欠です。機能が決定論的である従来のソフトウェアとは異なり、生成AIアプリは、現実世界の入力に遭遇すると、品質のドリフトや予期しない動作を示す可能性があります。効果的なモニタリングにより、問題を早期に検出し、使用パターンを理解し、アプリケーションの品質を継続的に改善することができます。

エージェントモニタリングの設定

Mosaic AI には組み込み [モニタリング機能](#) があり、カスタム モニタリング インフラストラクチャを構築せずにエージェントのパフォーマンスを追跡できます。

1. デプロイされたエージェントの [モニターを作成します](#)。
2. **サンプリングレートと周波数は**、トラフィック量とモニタリングのニーズに基づいて構成します。
3. サンプリングされたリクエストを自動的に評価するには、**quality メトリクスを選択します**。

主要なモニタリングディメンション

一般に、効果的なモニタリングは、次の3つの重要な側面をカバーする必要があります。

1. Operational メトリクス

- 。リクエストの量とパターン。
- 。応答のレイテンシ。
- 。エラー率とエラーの種類。
- 。トークンの使用量とコスト。

2. Quality メトリクス

- 。ユーザークエリとの関連性。
- 。取得したコンテキストでの接地性。
- 。安全性とガイドラインの遵守。
- 。全体的な品質の合格率。

3. ユーザーフィードバック

- 明示的なフィードバック (親指を立てる/下げる)。
- 暗黙的なシグナル(フォローアップの質問、放棄された会話)。
- サポートチャンネルに問題が報告されました。

モニタリング UI を使用する

モニタリング UI は、2 つのタブを通じて、これらのディメンションにわたる視覚化された知見を提供します。

- **チャート タブ**: リクエストの量、品質メトリクス、レイテンシ、エラーの経時的な傾向を表示します。
- **[ログ] タブ**: 個々の要求と応答を、評価結果を含めて調査します。

フィルタリング機能を使用すると、ユーザーは特定のクエリを検索したり、評価結果でフィルタリングしたりできます。詳細については、 [What is Lake House monitoring for GenAI](#)を参照してください。 (MLflow 2).

ダッシュボードとアラートを作成する

包括的なモニタリングのために:

- 評価されたトレーステーブルに保存されているモニタリングデータを使用して、**カスタムダッシュボードを構築します**。
- 重要な品質または運用上のしきい値の**アラートを設定します**。
- 主要な利害関係者との **定期的な品質レビューをスケジュールします**。

継続的な改善サイクル

モニタリングは、改善プロセスにフィードバックすることで最も価値があります。

1. モニタリング メトリクスとユーザーフィードバックを通じて **問題を特定します**。
2. **問題のある例** を評価セットにエクスポートします。
3. MLflow トレース分析と LLM 判断結果を使用して **根本原因を診断** します (「[一般的な品質問題とその修正方法](#)」で説明)。
4. 拡張された評価セットに対する **改善を開発およびテスト** します。
5. **更新プログラムをデプロイし、影響を監視** します。

この **反復的なクローズドループアプローチ**により、 エージェントは実際の使用パターンに基づいて改善を続け、変化する要件やユーザーの行動に適応しながら高品質を維持できます。Agent モニタリングを使用すると、本番運用におけるエージェントのパフォーマンスを可視化し、問題に積極的に対処して品質とパフォーマンスを最適化できます。

この記事は役に立ちましたか？

