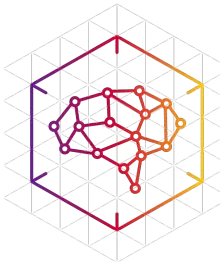
DATASTAX

ホワイトペーパー

生成 AI アプリのための ベクトル検索



AI アプリケーション開発にベクトル検索を活用するための開発者/アーキテクト向けガイド

このドキュメントは、生成 AI アプリケーションを企業独自のデータと組み合わせて、設計・構築しようとしている全ての方のためのガイドです。組織が理解すべき重要な概念と考慮事項を取り扱うだけでなく、ベクトル検索を用いて、LLM の持つ機能を大幅に拡張するためのシンプルで強力なアプローチについても解説します。

生成 AIの登場は、プロダクトを利用する方法だけでなく、プロダクトを構築する方法にも変化を与えています。OpenAIが2022年11月30日に ChatGPTを発表して以来わずか数ヶ月の間に生成 AIへの関心が世界を席巻しました。この記事を書いている時点で、マッキンゼーは、今や生成 AI の価値は世界経済全体で2兆4,000億ドルから4兆2,000億ドル規模になる可能性があると推定しています[1] 。

この革命の中心には、大規模言語モデル (LLM: Large Language Model) によって可能になったイノベーションがあります。LLM は急速にインフラストラクチャの重要な部分となり、主要なクラウド プロバイダーその他で採用されています。こうした最近の傾向を示すものとして、マイクロソフトのOpenAIへの投資[2]、Microsoft Azure OpenAI Serviceの立ち上げ[3]、グーグルのAnthropicとの提携[4] 、PaLM 2のローンチ[5]、Bard[6]、そしてHuggingface[7] のような数十のモデルを持つ盛んなオープンソースコミュニティ、Amazon Sagemaker[8] やグーグルのVertex AIと Generative AI App Builder[9] のような生成AIアプリを構築するためのツールとサービスの登場があります。

開発者はすでに新しい言語モデルスタックについて議論しています[10]。LangChain[11]やLlamaIndex[12] のような著名な新しいフレームワークが登場し、ベクトル検索が、生成 AIアプリケーション構築における技術スタックのもう 1 つの重要なコンポーネントとして浮上しました。ベクトル検索は、長期メモリ、プロプライエタリデータへのリアルタイムアクセス、生成 AI アプリケーションのカスタマイズに用いることができます。

生成AIへの関心の高まりを示すそのほかの事実と調査結果をいくつか紹介します。

- ChatGPTは、3か月も経ずに月間アクティブ ユーザー数1億人を史上最速で達成しました[13]
- OpenAIプラグインは、その発表以来、21の異なるカテゴリにわたる400を超える規模のエコシステムに瞬く間に成長しました[14]
- 開発者の92%が生成 AI ツールを使用しています[15]
- ベンチャーキャピタルの支援を受けている企業の90%が生成AI製品の発売を計画しています[16]
- 経営幹部の65%が、生成AIは組織に対して非常に大きな影響力を持つと考えています[17]
- SEMRush社は、ChatGPTを2023年の最も利用されたGoogle検索語の第26位にランクしました[18]

本稿では、生成 AI が重要な理由、生成 AI を効果的に機能させるためのアーキテクチャとデザインパターン、そしてベクトル検索が生成AIアーキテクチャの重要なコンポーネントとなった経緯について解説します。

生成AIの登場により、AIを活用したアプリケーションの構築は、かってないほど容易になった

生成 AIと大規模言語モデル（LLM）は多くの熱狂を生み出しています。それは、開発者にとって、AI アプリケーションの構築方法を劇的に簡略化させたことで、さらに革命的なものとなっています。その理由を理解するために、従来のアプローチと生成的なアプローチの違いを見てみましょう。

従来のAIアプリケーションの構築手順

生成AIが普及する前は、AIアプリケーションを構築する際に多くの障壁がありました。そのプロセスの主なステップは次のとおりです。

1. データから課題に関連する部分を特徴量として選別
   「課題に関連する部分」が何であるかを判断するのは難しく、しばしば、経験に基づいた推測の代わりに、

それを理解するためだけに別のモデルを構築する必要さえあります。生データから、課題に関連するデータを特徴量として選別することは、多くの場合、困難な問題であることが分かっています。たとえば、Netflixの最も重要な目標の1つとして、ユーザーが好むであろうコンテンツをユーザーのログイン時に予測することがありますが、このケースにおいても、特徴量は多くのデータセットにまたがっており、複雑を極めています。

2. 特徴量を使った要件に応じた機械学習モデルのトレーニング
画像分類などの一般的な機械学習の問題であれば、個別の問題に合わせて「ファインチューニング」することできる既製のモデルが存在する場合があります。Google の AutoML Vision[19] はそのようなシステムの好例です。既存のモデルに、犬がどのようなものであるかを示すサンプルデータセットを提供するだけで、画像に写った犬を認識することのできるチューニングされたバージョンのモデルが得られます。一方、ファインチューニングするための既存のモデルを見つけることができないようなタイプの問題があります。その場合、独自のモデルをトレーニングする必要がありますが、これは多大な時間を要し、計算コストが跳ね上がる傾向があります。一方、新しいユースケースに合わせて既存のモデルを変更する場面において、求める結果を導き出すためには、基礎となる数学に対する深い理解が必要になります。

3. モデルをデプロイし、**API** として公開
機械学習モデルをデプロイする際、アプリケーションが機械学習モデルを提供するサービスを利用できるようにすることになります。それには、モデルサービスが必要なデータに確実にアクセスできること、アプリケーションがモデルサービスにアクセスし予測ステップの結果をリクエストするためのインターフェースを設計することが必要です。このステップのAPI 実装の選択は、予測がどのように使用されるかに直接的に依存します。また、認証/認可についての配慮と実装も必要になります。

4. プロダクション環境でモデルを実行
モデルがプロダクションで利用され始めると、しばしば新たな課題に遭遇します。一般的な例としては、モデルの予測（出力）ステップのレイテンシ、時の経過に伴うモデルの陳腐化によるアウトプットの品質低下、モデルにとって重要なデータの投入までに生じる時間的ギャップなどが挙げられます。これらの問題のそれぞれを解決するために利用できるソリューションはありますが、ピーク性能を維持し、時間の経過とともに変化するデータや、データ量に対処するには、システムの長期運用ライフサイクルの様々な側面に繊細な注意を払い続ける必要があります。

モデルのデプロイと公開には一見特に複雑なところはないと思われるかもしれませんが、課題がないわけではありません。また、上記のプロセスのその他すべてのステップに渡って、しばしばオーダーメイドのソリューションが必要になります。さらに、従来のスタイルのAI アプリケーションを構築して出荷するために必要なスキル セットを持つチームを編成・維持するのは決して容易ではありません。達成するには、およそ3、4人からなるチームが必要になります。たとえば、従来の AI アプリケーション チームは、それぞれスキルの異なる、次のような役割で構成されるのが一般的です。

- データエンジニア - 使用する生のデータセットを取得し、探索と利用を容易にするためにデータを保存およびクリーニングする責任を持ちます。データエンジニアチームは、データモデリング、ETL パイプライン、データウェアハウジングの専門家集団です。
- データサイエンティスト - データを機械学習モデルで活用できるように~~ベクトル~~に変換する、データ探索と特徴量エンジニアリングを担当します。このチームは、モデルのソリューション化に際して、バイアスや過学習

などの問題を回避する必要があります。そのためには、AI モデルの適切な選択とトレーニングが重要となり、それには、しばしば統計学に関する深い知識が求められます。

- 意思決定科学の専門家（デシジョンサイエンティスト） - A/Bテストや仮説検証に関する専門家集団。このチームは、モデルがプロダクトに対して適正な成果をもたらすことを保証するようサポートします。また、AI 製品の構築に多額の投資を行う前に、データ サイエンティストが主導する AI モデル構築に先立つ探索的分析を支援し、データ内の洞察が単なる相関関係ではなく因果関係を示しているかどうかを判断する場合もあります。多くの企業では、データサイエンティストが、このデシジョンサイエンティストとしての役割も担っています。
- 機械学習エンジニア - 主に本番環境でモデルを実行する最後のステップを担当します。このチームは、特徴量エンジニアリングとモデル予測のステップが大規模実行に耐えること、適切に監視されていること、およびアプリケーションにとって許容可能な応答時間で予測が返されることを保証します。

これらはすべて、深い技術スキルが必要となる高度に専門化された役割であり、これまでAI アプリケーションを成功裡に構築して運用するためには必須とされていました。

生成AIを用いたAIアプリケーションの構築手順

LLM を使用した生成AIに誰もが興奮している理由の大きな部分として、上記の手順を踏まずに、特に以前は必要とされていた複雑なオーダーメイドのソリューションを使用せずに、問題を解決できることがあります。

生成AIにおいては、やるべきことは以下だけです。

1. LLM にデータを補完する方法を理解する（コンテクスト）
2. AIモデルに実行させたいタスクを自然言語で記述する (プロンプト)
3. LLM APIへのAPIコールを実行する

従来のAI アプリケーションに比べて、プロセスは劇的に単純になっています。そのため、生成AIを扱うことは、あらゆるチーム、あらゆる開発者にとって、十分現実的な範疇にあるといえます。

> 「最もホットな新しいプログラミング言語は英語だ」
>
> Tesla社のAI ディレクターAndrej Karpathy氏のツイッター上での発言[20]

生成コンテキスト

生成AIアプリケーションを構築する際の最も重要なステップは、LLM にコンテキストを提供する最初のステップです。これは、検索拡張生成（**RAG: Retrieval Augmented Generation**）と呼ばれます。 追加のコンテキスト情報で補うことがなければ、LLM は、確かに「広く」使えはしても、ビジネスの急所について「浅く」しか捉えることができず、要は単なる興味深いおもちゃ以上のものにはなり得ません。 RAGは、LLMに実際にビジネス上の問題を解決する力を与えます。

しかし、LLM に対して、いくら情報を与えたところで、それらが課題に関連する情報でない場合、役には立ちません。意味的に課題に関連する情報（*semantically relevant information*）を与えることが重要になります。ベクトル検索を使えば、そのような情報を特定することは驚くほど簡単になります。

ベクトル検索とは何か?

ベクトル検索は、類似した属性または特徴を持つ関連オブジェクトを検索する方法です。ベクトル検索を活用することができるオブジェクトの一般的な例には、テキスト、物理的な製品、画像、ビデオなどがあります。

ベクトル検索では、エンベディングと呼ばれる特定のタイプの機械学習モデルを使用して、オブジェクトとそのコンテキストを記述します。エンベディングされたデータは、オブジェクトのセマンティクスを捉えるベクトルであり、次のセクションで詳しく説明します。このアプローチにより、ベクトル検索では、正確にどの部分が類似する対象を探すことになるかについて前知識を持つ必要がなく、類似したデータを見つけることができます。

英語を例に挙げると、「happy」、「cheerful」、「joyful」という単語はすべて同様の意味を持ちますが、従来のキーワードベースの検索では、「happy」というキーワードを用いた場合、「cheerful」と「joyful」に一致するドキュメントは検索されません。これを解決するのが、ベクトル検索の能力です。ベクトル検索は意味を理解することができるため、ユーザーが網羅的に指示せずとも、検索したい対象を伝えることができます。

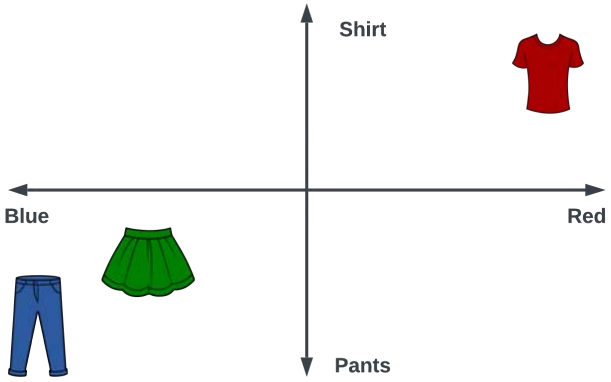
もう一つの利点として、ベクトル検索は、近似最近傍 (ANN) 検索と呼ばれるアルゴリズムのセットを使用して非常に高速に実行できることがあります。このアプローチは非常に効率的であり、数学的に互いに類似するベクトルを素早く導出します。

エンベディング（埋め込み）の基礎

エンベディング（埋め込み）は、テキストの意味を封じ込めたデータの数学的表現です。テキストをベクトル化して埋め込み（エンベディング）表現に変換するために、ドキュメントを構成する単語群から数値のリストへの変換を実施します。エンベディングの結果、ベクトル空間内では、互いに関連性の高い意味を持つベクトル同士は、近接した場所に配置されることになります。

もっと分かりやすくするため、例を用いて考えてみましょう。図 1 は、「衣服の種類」に関する次元と「服の色」という属性に関する次元の2つの次元を持つ単純なベクトル空間を示しています。

図1. 衣服を表現する2次元ベクトル空間

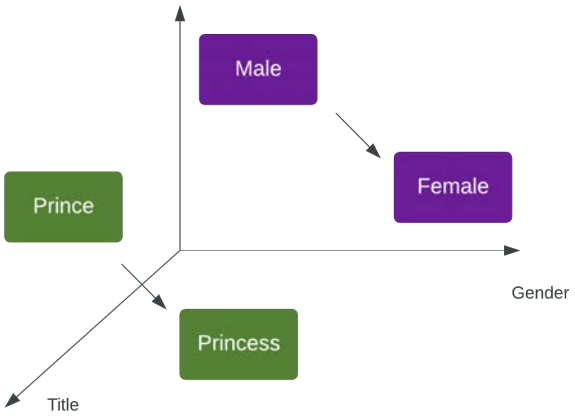


この例では、ベクトルは2次元の情報を持ち、ベクトルデータを構成するそれぞれの要素は、次の2つの情報を表してます:[<衣服の種類>, <服の色>] 。ここで、「赤いシャツ」のベクトルは [1, 1]で、「青いズボン」のベクトルは [-1, -1] で表されているとします。そこに、「緑色のスカート」を加えます。スカートは機能的にはズボンに近く、緑は色のスペクトルとして青に近いため、そのエンベディングベクトルは「青いズボン」に近い位置にあるはずで、例えば [-0.8, -0.8] のような値を取ると推測することができます。

上の例は、エンベディングを使ってベクトル空間を構築する方法と、その空間の中で、オブジェクトが存在する場所についてのアイディアを示しています。

次に、エンベディングに対する算術演算がどのように機能するかを示す例を見てみましょう。予め注意しておくと、ベクトル空間は実際には非常に大きな次元を持つ(ベクトルは多くの数値で構成される)のが一般的です。ここで扱うベクトル空間は、「人物の肩書き」と「性別」という二つの概念を捉えた情報を扱っています。図2を参照ください。

図2. セマンティック情報を保持するベクトル演算の例



埋め込み（エンベディング）表現はベクトルであるため、ベクトル同士で算術演算を行うことができます。その際、その演算結果のベクトルには、ベクトルの元となった自然言語の組み合わせに相当する意味概念が保持されることが期待されます。

たとえば、ベクトル同士の計算において、《「王子」 － 「男性」 ＋ 「女性」》という計算式の結果として得られるベクトルは、「プリンセス」を表すベクトルと類似する値であると考えられます。

同様に、《「王子」 － 「男性」》と《「プリンセス」 － 「女性」》という２つの計算式の結果のベクトルは、互いに非常に近い距離にあるでしょう。この場合、これらのベクトルは、「王または女王の直系の子孫である王族（男女の別を問わない）」という概念を捉えていると見ることができます。

検索拡張生成（RAG）パターン

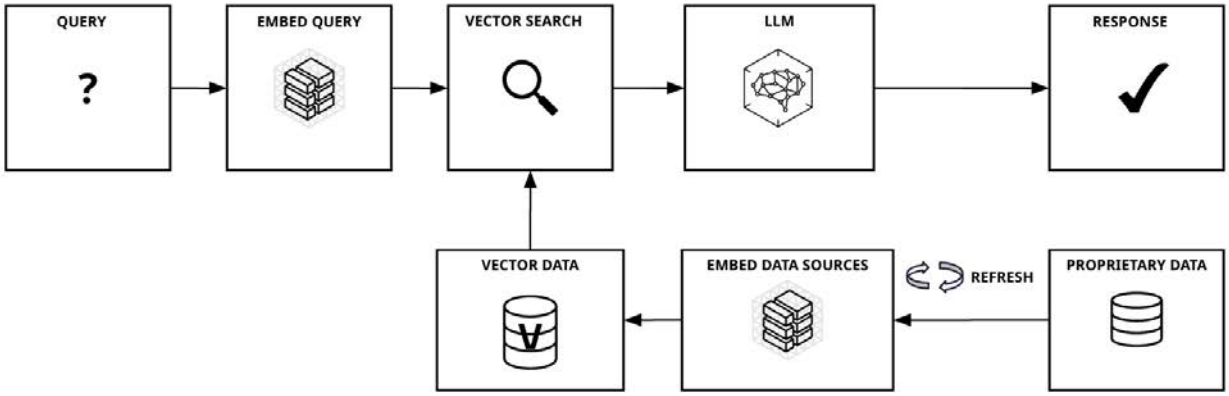
LLMには、それが素のままで用いられた場合、生成された応答が特定のアプリケーションでは用をなさない可能性がある、という限界があります。これは、LLM に保存されている知識が通常そのモデルのトレーニングに用いられたデータに限定されており、インターネット上で一般に入手できないような、最新のデータや、特定のドメインに関連するプロプライエタリデータが含まれていないことに由来します。

検索拡張生成（RAG）の中心となるアイディアは、LLM と、ユーザーが実行を依頼するタスクの解決に関連する独自のドキュメントとを統合することです。

図3は、RAGを用いたアプリケーションがどのように動作するかを示しています。まず、クエリがアプリケーションに対してサブミットされます。生成AIアプリでは、クエリの表現は、検索エンジンタイプ、つまり「ジョージ・ワシントンの歴史」というようなキーワード、または「ジョージ・ワシントンはいつ生まれたか」のような具体的な質問文、さらには「ジョージ・ワシントンの生涯についてのエッセイを書け」という命令文、といったように様々な形式をとり得ます。独自データソースのエンベディングに使用したものと同じエンベディングモデルを使用してクエリのエンベディングを行い、そのクエリのエンベディングを使用してベクトル検索を行うことによって、そのクエリに最も類似するデータを検索することができます。このようなデータ間の類似性に基づく効率的な検索は、ベクトル検索によって実現される重要な機能です。

最後に、アプリは検索に該当した上位数件のデータのテキストを取得し、それらをLLM に与える命令文の作成に活用して、LLMにタスクの結果を生成させます。

図3. RAGパターンを用いたLLMアプリケーションの基本フロー



このアプローチには、次の利点があります。

- ハルシネーションの軽減
  AIの分野では、一見すると尤もらしく見えるものの、事実とは異なる、あるいは文脈に適さない回答が出力されることを「ハルシネーション（Hallucination：幻覚）」と呼びます。RAGデザインパターンは、ベクトル検索によって抽出されたドキュメントに焦点を当てるように LLM に指示して、応答を生成します。LLMのトレーニングに用いられたデータセットの全体ではなく、より文脈に適したドキュメントセットを使用することで、クエリへの応答は、ユーザーの意図をより正確に汲み取ったものになります。

- プロプライエタリデータの活用
  企業がLLMアプリケーションを開発する場合、LLMの生成する応答を、その企業の製品・サービスとユースケースに対応した内容にするために、そのアプリケーションの要件に関連した企業の保有する情報・ドキュメントを、LLMに直接提供することができます。

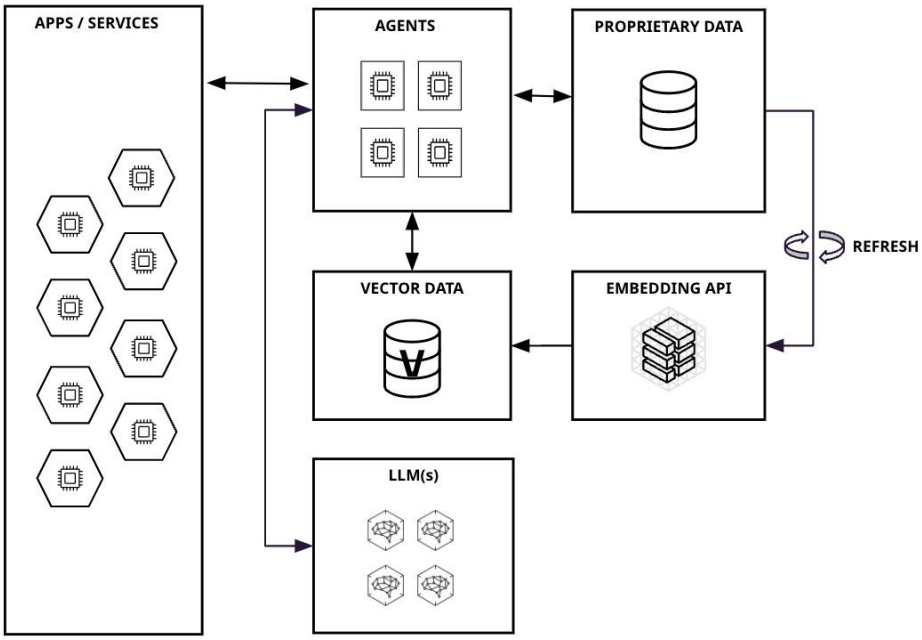
- **LLM** 「トレーニング」の簡便さ
  RAGパターンは、特定のアプリケーションに合わせてモデルを調整する方法の中でも、実装および保守が容易です。プロプライエタリデータへの新しいドキュメントの追加、既存のドキュメントの変更、または新しいデータソースが利用可能になったときにも、エンベディングデータを更新して、モデルを最新の状態に、そしてその応答を正確なものに保つことが容易です。

次の図3.1は、RAGを使用したアプリケーションまたはサービスを実行するための完全なアーキテクチャの概要を示しています。このアーキテクチャでは、エージェントという新たな概念が導入されています。エージェントとは、自動化された推論・意思決定エンジンです。エージェントは、ユーザーからの入力すなわちクエリを受け取り、それに応じて何を行うかを決定するアプリケーション・コンポーネントまたはプログラミング・コードです。

図3で紹介した単純なRAGの例では、エージェントのタスクは、ユーザークエリのエンベディング、ベクトル検索の実行、検索結果を含むLLMのプロンプトの作成といった一連の基本的なプロセスからなっていました。

より複雑なアプリケーションでは、タスクを複数のサブタスクに分割した上で、問題を解決するために外部ツールを使用する必要があるかどうかを判断したり、アプリケーション用のメモリを使ってデータを保存し維持するために、エージェントが利用される場合があります。

図 3.1 RAGパターンを用いたアーキテクチャのコンテキストビュー



ここで、エンベディングデータの保存とベクトル検索の実行にベクトル対応データベースを採用すると、利用可能なデータ量を非常に大規模なデータセットに拡張しながら、ドキュメント検索速度の性能を維持できるという利点を得ることができます。ベクトル対応データベースは、あらゆる生成エクスペリエンスの基本要件である、類似性検索を高い効率で実現する機能を備えています。

ベクトル対応データベースの採用

Sequoia Capital 社の最近の調査[10]によると、生成AIに取り組む**88%**の企業が、生成AIアプリケーションの実現に向けて、ドキュメント検索メカニズムが、今後インフラストラクチャを構成する重要な部分になると考えています。

RAG AIアプリケーションの実装

検索拡張生成（RAG）を用いた生成AIアプリを構築するための、基本的なフロー（図 3）とアーキテクチャダイアグラム（図 3.1）を紹介しました。ここでは、それらをさらに詳しく見てみましょう。

RAGシステムを構築するには、次の2 つのワークフローが必要です。

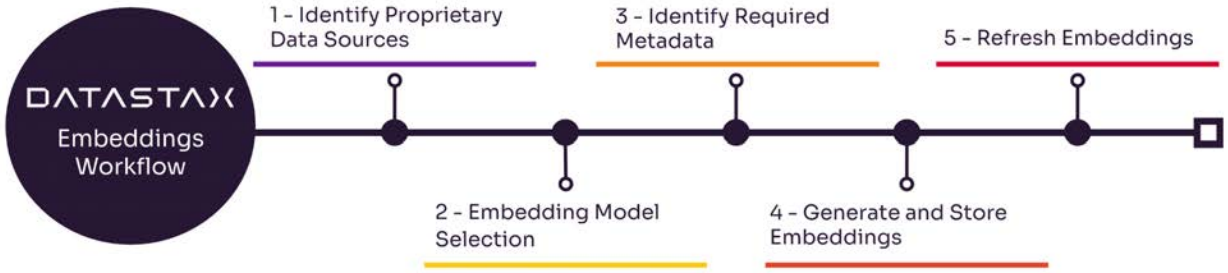
- ナレッジベースの作成
- 生成プロセスの実行

以下、それぞれについて見ていきます。

ナレッジベースの作成

まず、RAGアプリケーションがアクセスする、エンベディングデータからなるナレッジ ベースの構築について見てみましょう。このワークフローには複数のステップがあり、その概要を図4に示します。

図4. データセットをベクトルストアに変換するワークフロー



1. プロプライエタリデータの特定

   初めのステップは、どのようなアプリケーションを構築しているかによって異なってきます。まず、アプリケーションで使用できそうなデータセットを検討することになりますが、その際には、そのデータソースはどのような固有の情報を提供するか？、エンベディングに利用できる公開データセットはあるか？、といった点を検討する必要があります。

   公開データセットの中には、LLM構築時の トレーニングに用いられたデータセットよりも新しいデータが含まれている可能性があり、新たに、そうしたデータセットを用いることも考えられます。たとえば、最近のニュース記事や最新の開発者用ドキュメントといったものがあります。

2. エンベディングモデルの選択

   エンベディングモデルの選択において、ナレッジベース用のエンベディングモデルと最終的な応答のテキス

トを生成するLLM内で採用されているエンベディングモデルは、同じである必要はありません。また、ナレッジベース用のエンベディングモデルとLLMを、同じベンダー（またはオープンソースのモデルプロバイダー）から選択する必要さえありません。ただし、ナレッジベースを構成するデータセットの作成と、そのデータセットに対するクエリの両方に、同じエンベディングモデルを用いることが重要です。

エンベディングモデルの選択を行う際に、様々なモデルの性能の違いについての見通しを得るために適した情報源として、「大規模テキストエンベディング・ベンチマーク（Massive Text Embedding Benchmark, MTEB）[21] 」があります。ただし、このベンチマークを用いる際には、ベンチマークデータは定期的に再計算されており、計算途中の場合には、完了するまでリーダーボードが利用できないことに注意してください。このような時には、時間をおいて、もう一度確認してください。

取り敢えず始めてみたい場合には、OpenAI のtext-embedding-ada-002[22] モデルはAPIが公開されており、性能も良好です。

自らホストすることのできるオープンソース技術を利用したい場合、Instructor[23]モデルは優れた性能を備え、ユースケースに合わせてエンベディングを調整することもできます（Instructorモデルには3 つのサイズがあり、小さい方の2つのモデルは、ラップトップでも十分実行可能です）。

3. メタデータ利用の検討

ベクトルデータそのものの利用だけでは十分でなく、ベクトルデータにメタデータを関連付けることが必要になる、いくつかの理由があります。

まず、ベクトル検索で一致する上位数件を取得した後、LLM に渡すプロンプトでは、それらのエンベディングデータを、通常のテキストとして利用することになります。

エンベディングは通常、チャンクと呼ばれる文書全体よりも短いテキストのセクションに対して行われます。ユースケースやデータセットの種類によっては、各チャンクと一緒にドキュメントの全文を保存することが有効な場合があります。この判断には、データセットそのものに関する知識が必要です。ドキュメントが非常に長く、多くのトピックについて説明している場合は、チャンクのテキストを使用する方が、それぞれのチャンクが別のクエリへ関連する可能性が高く、より合理的です。一方、文書がすべて特定のトピックに関するものであることがわかっている場合、完全な文書を利用する方が、LLMにとって有益な詳細情報が含まれる可能性があります。

その他の種類のメタデータとして、検討の価値があるものとして、そのデータを一意に識別するためのIDがあります。ドキュメントIDは、ソースドキュメントの変更によりエンベディングデータを更新する必要があるかどうかを確認するときに利用できる場合があります。

ベクトル検索を実行したいデータセットのサブセットを選択するためにフィルターできるメタデータを追加することも役立つ場合があります。簡単な例としては、「製品ドキュメント」や「社内 Wiki」など、さまざまなデータソースのカテゴリー情報を利用することが考えられます。 例えば、「社内 Wiki」には特定のユーザーと共有したくないデータが含まれている場合があるため、そのソースを検索結果から除外する、といった風にこの

カテゴリー情報を利用することができます。

4. エンベディングデータの生成と保存

先に、エンベディングに用いるためにモデルを選択しました。このモデルを用いてベクトルを生成することになります。

このプロセスの最初のステップは、テキストのチャンク化 です。これは、テキストをより短い文字列に分割する方法を決定するプロセスです。これは主に次の2つの理由から行われます。

まず、エンベディングモデルには入力として受け入れることのできるテキストの量に制限があるためです。text-ada-002 の場合、その制限は 8,192 トークンです。受け入れられる文字列の長さがより短いモデルもあります。

また、チャンク化要否の判断は、モデルが持つ制限以外にも、データ ソースとアプリケーションの両方の面からも検討する必要があります。アプリケーションがドキュメント全体をユーザーに返す必要がある場合、モデルが許す限り、できるだけ大きなチャンクを用いることは合理的です。あるいは、アプリケーションが、例えば質疑応答システムの拡張といった場合のように、様々な事実に依存している場合、ドキュメント全体よりも、その中の数センテンス程度の短いチャンクの方が、一つのトピックに関する事実やアイデアを扱っている可能性が高く、このユースケースにより適しています。

さて、チャンクの長さが決まったら、次はチャンク化に関する戦略を立てる必要があります。これには多くの方法があります。

最も簡単な方法は、テキストを均等な長さのチャンクに分割することです。この場合、重要な事実を説明している途中でチャンクが単語や文を分割してしまうことが生じる可能性があるため、チャンクのオーバーラップを許容すると文書内の概念を適切に捉えるのに役立ちます。

より高度な、広く用いられている戦略として、ドキュメントの、または言語の構造に基づいたチャンク化があります。例としては、ドキュメントのセクション、段落ごと、またはセンテンスの境界を基準にしたチャンク化が挙げられます。

ドキュメントのチャンクのセットを作成したら、各チャンクのベクトルを、先の手順で定義した関連メタデータと共に、データ ストアに保存します。これにより、近似最近傍探索（ANN: Approximate Nearest Neighbor）を利用した高速ベクトル検索が可能になります。 DataStaxのApache Cassandraマネージドサービスであ るAstraDB は、ベクトル検索の機能を有しており、大規模なベクトルデータセットを操作する場合の優れた選択肢となります。そしてまた、AstraDBのベクトル検索機能は、高い可用性と柔軟なスケールを有した分散データベースApache Cassandraの拡張機能であるため、ベクトルとともに、そのベクトルに関連するメタデータを保存しておくことも容易であり、ベクトル検索のみではなく、メタデータを使った操作との組み合わせについても、利点を持っています。このような大規模データ利用のための信頼性・スケールと、データベースとしての汎用性は、AstraDBの持つ、単なるベクトルデータベースとは異なる利点です。

5. エンベディングデータの更新

企業が保有するデータは、時間の経過とともに変化していきます。ハルシネーションを避けるためには、新しいデータソースが利用可能になったらすぐに、エンベディングデータを更新することが重要です。

データ ソースの変更が発生するのは、ある時は、新しいドキュメントが利用可能になったためかもしれませんし、またある時は、過去にエンコード済みのコンテンツが更新されたためである場合もあります。

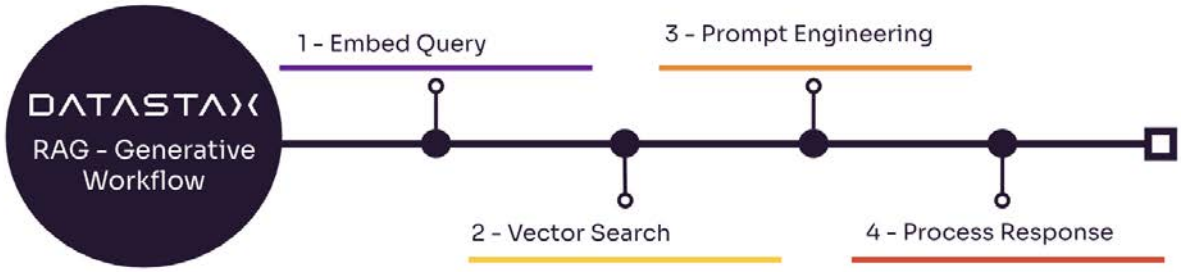
既存のドキュメントのエンベディングデータを更新するときは、企業が保有するデータの1オブジェクトに対して複数のエンベディングが登録されている可能性があることに注意する必要があります。データソースがテキストである場合、これはテキスト全体の長さと選択したチャンク化の戦略によって異なります。エンベディングデータを更新するときは、更新時にそのドキュメントから派生した、すべてのエンベディングデータを置き換えるように注意する必要があります。

また、すべてのエンベディングモデルの変換ロジックが決定論的であるわけではないことを覚えておくことは有益です。モデルによっては、同じテキストをエンコードする都度、異なる結果が得られる場合があります。結果として得られるベクトルは通常、類似したものにはなるでしょうが、既存のエンベディングデータを新しいエンベディングデータに置き換えるときには、置き換えるべき既存のエンベディングデータの特定のために、エンベディングベクトルの生の値を用いることは得策ではありません。代わりに、一意の識別子を使うことが考えられます。

生成プロセスの実行

ここからは、前のセクションでエンコードしたデータセットを使用して LLM からの応答を生成する方法を詳細に見ていきます。

図5. クエリから応答を生成するワークフロー



1. クエリのエンベディング

RAG アプリケーションにおけるテキスト生成はクエリから始まります。クエリはアプリケーションのエンドユーザーから送信されたものを、そのまま利用する場合もあれば、それを元にLLM またはその他の手続き型コードによって新たに生成する場合もあります。

次に、クエリのテキストエンベディングを行って、クエリをベクトルにエンコードします。

ここで使用されるエンベディングモデルが、ベクトル検索用のデータセットをエンベディングするために使用されたものと同じモデルであることが重要です。そうでない場合、ベクトル検索は正しく機能しません。

2. ベクトル検索

次に、ベクトルデータストアからクエリに最も類似したエンベディングデータを取得することになります。まず、クエリベクトルの近くにあるベクトルを探すため、最近傍検索を行います。多くの場合、この検索は「上位N件」の一致を求めることによって行われます。例えば、上位 3 件の一致を利用する、というようなことが一般に行われますが、この数は、エンベディングで使用したチャンクの長さ、コンテキストウィンドウ（LLM に許可される入力のサイズ）、およびユースケース等に応じて調整することになります。

ベクトル検索を行う際に、検索結果のみでなく、クエリベクトルと検索結果ベクトルとの間の類似性についてのスコアまたは尺度を返すように要求することがあります。このスコアがしきい値を下回る場合、検索されたドキュメントのクエリに対する一致度が十分なものでないとして、後続のプロセスで、この検索結果ドキュメントを、次のステップでは利用しないという決定を行うことができます。使用する実際のしきい値は、アプリケーション、データセット、および類似性メトリックによって異なります。スコアの範囲が、最大値 を「1」として正規化されていると仮定する場合、カットオフの値として、「0.5 〜 0.8」の間から実験を開始するのが適切と言えるでしょう。

ベクトル検索により、 たとえば 数百、数千件規模の一致結果を取得すると共に、それぞれのスコアを得た上で、エンベディングの生成に使用された元テキストを使って選別を行うことで、適切なカットオフを、ベクトルデータストアに任せるのではなく、アプリケーションが決定することができます。結局のところ、ベクトルストアから数学的に導き出された結果に対して、クエリの本来の目的との関連性が保たれている地点を探すことが重要です。

クエリとの関連性の高い結果ベクトルのセットを取得したら、次のプロンプトエンジニアリングのステップで使用するために、それらのベクトルとともにメタデータとして保存されている関連テキストを取得します。

3. プロンプトエンジニアリング

プロンプト エンジニアリングとは、LLM に与える命令を設計することです。一部の LLM は、プロンプトを1つだけ受け取ります。あるいは、チャットモデルの場合に典型的ですが、命令は、プロンプトとして定義する必要がある複数の異なる入力項目で構成されている場合があります。このようなプロンプトの各セクションについて、以下で個別に説明します。

1つ目はコンテキストです。コンテキストは、モデルに何を行わせたいかを表します。例えば、「このコードについて説明しなさい」や「カスタマーサービスボットとして質問に答えてください」等です。

加えて、LLMに対して、いくつかの回答のサンプルを与える場合があります。これはFew-shot学習と呼ばれ、LLM に、ユーザーの意図をより良く理解させるのに役立ちます。チャット用LLM の場合は、通常、サンプルとして会話の両サイドの情報を提供する必要があります。具体的には、ユーザーからの質問と、AIから期待する応答の両方をテキストサンプルとして与えます。

RAG アプリケーションでは、LLM への命令の中に、取得済みのベクターエンベディングのテキスト表現を挿入することになります。このパターンでは、次のような形式のインストラクションがLLMに対して与えられます。

> "Use the articles below to complete the task.
>
> **Article:**
> <ベクトル検索で最初に一致したテキストを挿入>
>
> **Article:**
> <ベクトル検索で2 番目に一致したテキストを挿入> "

ベクトル検索で一致したテキストを、含めたいだけ「Article」セクションに追加することができます。さらに、タスクの指示として、上記例のような一般的な表現（"complete the task"）ではなく、より具体的な記述を用いることができることを忘れないでください。質疑応答システムの場合であれば、例えば、「以下の記事の事実を、質問に答えるために使用しなさい」というコンテキストを用いることができるでしょう。

また、より実践的な改良として、ベクトル検索によって選ばれたドキュメントを使って回答を生成することを指示するだけでなく、そもそも、それらのドキュメントが依頼したタスクに関連しているかどうかについても推論するように、LLM に対して異なるレベルの指示を同時に与えることもできます。
たとえば、質疑応答システムでは、先に示したLLMに対する指示の表現を次のように変更できます。「以下のドキュメントの事実を、質問に答えるために使用してください。ただし、与えられたドキュメントが質問に答えるために有益でない場合は、『わかりません』と言ってください。」

結局のところ、LLM に実行を依頼するインストラクションまたはタスクを、ユーザーの意図が明確に示されたものとすることが重要です。

4. 応答プロセス

ベクトル検索プロセスによって選ばれたテキストの入力を含む、完全なインストラクションを提供すると、LLM から最終的なテキスト出力を受け取ることになります。ほとんどの場合、ここで、後処理が行われるこ

とになります。それは例えば、AI安全性分析、品質管理解析、そしてLLMからの応答に対してアプリケーションのユースケースに固有の付加的なフィルタリングやアジャストメントなどを組み込むというものです。多くのシステムでは、この後処理のステージを、生成された応答を、後で集合的にレビューするために、そのコピーをシステムに保存する機会としても使用します。LLM応答の後処理が完了すると、最終的な応答がユーザーに返され、一連のRAG プロセスの全体が完了します。

使用するLLMの選択

テキスト生成に使用できるLLMは多数あり、その中から利用するものを選択する必要があります。最小限のセットアップですぐに使い始めたい場合は、OpenAIモデルが良い出発点となるでしょう。OpenAIには、様々な利用可能なライブラリが用意されており[24]、Python、Node.js、そして、Azure OpenAIライブラリも利用可能です 。また他の多くの言語用のコミュニティ版のライブラリも存在します。

OpenAI モデルの中では、gpt-4 が最も強力で最高の結果が得られますが、すべてのアカウントで利用できるわけではありません。 gpt-3.5-turbo モデルも非常に強力で、はるかに安価です。コストをコントロールしながら構築を進めたい場合、このモデルは多くのプロジェクトの良い出発点になるでしょう。

上記のどちらのモデルにも、サイズの異なるコンテキストウィンドウを備えた 2 つの異なるバリエーションがあります。より大きいコンテキストウィンドウは、より多くのデータをプロンプトの入力に提供でき、より長い文書入力に耐え、ベクトル検索ステップで得られた文書サンプルをより多く利用することができます。一方、モデルのコンテキストウィンドウが大きいほど、1トークンあたりのコストも高く設定されています。そして、入力が大きくなると、請求されるトークンの数も増えるため、大きなコンテキストウィンドウを実際に活用する場合、コストは大幅に上昇します。そのため、アプリケーションに、そのような追加入力スペースが必要かどうかを判断することが重要です。

Google Cloud ユーザーの場合は、Vertex AI[25]で利用できるPaLM2モデル[5]があり、テキストとチャットで利用できます。オープンソースの選択肢については、Hugging FaceによるOpen LLM Leaderboard [7] をご覧ください。

ユースケース

セマンティック検索

セマンティック検索は、しばしば「意味や理解を伴う検索」と説明されます。それは、単語や語句を正確に一致させようとする語彙検索やキーワード検索とは対照的です。

セマンティック検索の威力を伝える、いくつかの簡単な例を示します。「ウェブサイトのテーマ」、「ウェブサイトのテンプレート」、および「ウェブサイトのデザイン」というフレーズはすべて、ウェブサイトの視覚的なプレゼンテーション（見た目）を表現しています。ただし、従来の語彙検索で「テーマ」というキーワードを使って検索すると、特定の製品がそれらを「テンプレート」と呼んでいる場合、そうした情報は検索結果として返されません。あるいは、画像の検索にセマンティック検索を用いる場合、「ハヤブサ」と「タカ」は、どちらも鳥であるため、ベクトル空間の近接した位置に配置され、お互いに検索が可能である一方、これらは「家」の画像とはかけ離れたものとして扱われます。

セマンティック検索で解決できる問題の種類は数多くあります。いくつかの例を次に示します。

- 検索エンジン
  おそらく疑問を挟む余地のないところとして、セマンティック検索の技術を用いて構築された検索エンジンは非常に強力かつ高速なものになります。

  この場合、単純にベクトル検索を実行して、検索結果の上位N件のデータを返します。ここで、N は返したいドキュメントの数です。あるベクトルに類似したベクトルの検索には、通常、近似最近傍（ANN）検索が用いられます。

ここで、より意義深い結果をユーザーに提示するために、利用可能な他のデータに基づいて検索結果を再ランク付けすることも考えられます。例えば、ドキュメントへのトラフィック、クリックスルー率、その他の指標を用いることが考えられます。

- ドキュメントの要約と情報抽出
  ドキュメントの長大なリストを提示し、ユーザーにそれらを一つ一つ読ませるのではなく、ベクトル検索で見つかったドキュメントをLLMを用いて要約したり、特定の情報を抽出した上で、ユーザーに提示することができます。

- レコメンデーション
  レコメンデーションシステムは、パーソナライゼーションのための強力なツールです。ユーザー毎に、そのユーザーが興味を持ちそうなコンテンツを確実に提示することで、システムによる顧客エンゲージメントの向上を実現することができます。生成AI本来のパワーと、プロプライエタリデータを用いて強化された汎用性との組み合わせにより、レコメンデーションシステムはさらに強力かつ広範囲をカバーするものになります。

  レコメンデーションの実装には様々な方法がありますが、ベクトル検索を使用するアプローチのごく基本的な概略は次のとおりです。

  - アイテム・エンベディングを生成します。ここで、アイテムとはユーザーに対する推薦の候補となる記事、ビデオ、写真、製品などです。アイテム・エンベディングは、コンテンツまたは製品のプロパティを内包します。これには、コンテンツであれば、ジャンルやトピックなどの属性、物理的な製品であれば、色、サイズ、スタイルなどの属性が含まれます。あるいは、コンフィギュレーションパラメータ、コンビネーションルール、共通コンビネーションが、プロダクト、サービス、オファリングのために利用される場合もあります
  - ユーザー・エンベディングを生成します。ユーザー・エンベディングは、それぞれのユーザーの好みを表現しているエンベディングベクトルです
  - ベクトル検索を使用して、ユーザー・エンベディングに最も似通ったアイテム・エンベディングを検索することで、レコメンデーションを生成できます

チャットやカスタマーサポート

生成AIによって、チャットやカスタマーサポートアプリを拡張することができます。これによって、ユーザーの質問に対してあたかも人間が行っているかのような応答を行うことが可能になります。これによく似たアプリの例として、バーチャル家庭教師やパーソナルアシスタントなどがあります。

重要なのは、同様のアプローチを使用して、幅広い付加機能を提供できるということです。例えば、クエリ、またはダイアログでの会話のやりとりの中で明らかになった、ユーザーの嗜好性に基づく推薦を行うことが考えられます。たとえば、高齢の肉親のために最適な老人ホームを探そうとしているユーザーは、様々な要件や制約を持つことでしょう。アプリを適切なエンベディングデータで拡張することによって、RAGベースのエクスペリエンスによって、可能な候補の中からさらなる絞り込みを行った上で、ユーザーにリストを提供することができれば、ユーザーは選りすぐられたリストから検討を始めることができます。

様々な場面で、このような体験を実現することは容易です。なぜなら、生成AIモデルは、従来のチャットやカスタマーサポートシステムとは異なり、事前定義された回答や構造化データベースに依存することがなく、質問のコンテキストとセマンティクスを理解した上で、あたかも人間のような応答をリアルタイムで生成できるからです。

このパラダイムをさらに一歩進めて、上述のようなエクスペリエンスによって生成された応答の履歴をエンベディングデータに変換した上で、将来の使用のために保存しておくことが考えられます。この追加機能は、RAGアプリケーションのためのデータプラットフォームを拡張して容易に実現可能です。この拡張は、単純なチャットボット履歴の保存と比べて、はるかに強力です。なぜなら、すべての顧客とのやり取りの長期記憶が、完全に検索可能なものとして構築され、将来の会話のコンテキストで参照できるようになるからです。

チャットやカスタマーサポートシステムは、先に説明したRAGデザインパターンを使用して LLM を用いて実装することができます。セマンティック検索の中心概念は、尋ねられた質問への回答に役立つドキュメントを見つけることです。これは、上述した検索の使用例ととても似通っています。ただし、チャットやカスタマー サポートシステムの場合には、さらに、セマンティック検索の結果を使用してクエリに対するソリューションを提案するように LLM に指示する後続の手順があります。この機能の組み合わせにより、エクスペリエンスははるかに汎用的になり、一様でない要求を持つユーザーにとって魅力の大きいものになります。

また、LLM を用いて質問と回答のペアを生成することもでき、生成された質問と回答のペアをアプリケーションがユーザーに提供する回答の精度向上のために活用することも考えられます。例えば、既存の文書から質問と回答のペアを生成するよう LLM に指示することができます。あるいは、既存の質問と回答のペアを、異なる方向に拡張することもできるでしょう。また、既存のナレッジベースでカバーされていないトピック領域をカバーするより多くの質問と回答のペアを含む合成データを生成することもできます。

分類

分類問題には多くのユースケースがあります。画像解析の分野は、その典型の一つです。例えば、写真に自動車が写っているか、画像に信号機が含まれているか、といったことが問われます。

テキスト解析における分類問題として、ドキュメントのトピック分類があります。例えば、ドキュメントの内容がスポーツ、政治、エンターテインメント等、どのジャンルに関するものであるかを判断します。

以下、LLM を使用した分類に対するいくつかの異なるアプローチと、それぞれのメリットとデメリットについて解説します。まずは、実装の面で最も簡単なものから始め、より複雑なアプローチへと進んでいきます。

分類のための**Few-shot**学習

LLMは、Few-shot学習プロンプトを用いて、テキスト分類のためにそのまま使用できます。Few-shot学習のプロンプトでは、分類用のさまざまなラベルの例をLLMに提供します。

ポジティブな感情とネガティブな感情を分類する簡単な例を見てみましょう。ここでは、OpenAI プラットフォーム 上で、gpt-3.5-turboモデルを使用して、これがどのように可能になるかを例示します。

Few-shot学習の例を示すために、まず、プラットフォームが提供しているタスク定義機能の使用例を示します。

[システムプロンプト]

（ここで、文章を肯定的なものと否定的なものとに分類しています）

Document:
This movie was horrible.
Sentiment:
Negative.

Document:
I loved the movie I saw this weekend, it was so exciting!
Sentiment:
Positive.

Document:
Worth every penny! It was soooo good!
Sentiment:
Positive.

これで、ユーザー プロンプトから新しいドキュメントを提示し、LLM にそのドキュメントに内在するセンチメントを返すようにリクエストする準備が整いました。

[ユーザープロンプト]

Document:
I hated that movie, it was so boring.
Sentiment:

この場合、LLM からの応答は「**Assistant: Negative**」になります。

分類のための**LLM**ファインチューニング

ファインチューニングは、一部のLLMでサポートされているオプションです。ファインチューニングでは、モデルに大規模な入出力ペアを提供し、目的に合わせて、モデルに実行させたい内容をチューニングします。 LLM の動作をファインチューニングするには、通常、少なくとも数百以上のサンプルを提供する必要があります。一般に、このように大量のサンプルをFew-shot学習用のプロンプトから渡すことは、入力テキストサイズの制限や、トークンに掛かるコストの面から、現実的ではありません。

以下では、Few -shotプロンプトでは役不足であり、ファインチューニングが必要となる場合について、考えます。例として、特定の製品がテキスト内で議論されたかどうかをラベル付けする分類の問題を検討します。Stripe社（金融

サービスを提供するSaaS企業）は、Payments、Checkouts、Billingなどの製品を提供しています。ご覧の通り、製品名には一般名詞が使用されています。このような場合、LLMにとって、固有名詞としての製品名と、一般名詞としての会計用語とを区別することは難しいものになります。

ここで、「My stripe payments failed.」というフレーズを考えてみましょう。

ユーザーは、「Stripe Paymentsをセットアップしようとしたが、うまくいかなかった」と言っているのかもしれませんし、仔細に表現すると「Stripeからの請求書を支払おうとしたが、失敗した」という内容を伝えようとしていたのかもしれません。

この単純な例では、フレーズを「Stripe Payments」という製品に関する議論として分類すべきかどうかを判断することは困難に見えます。

対策として、文書内で Stripe paymentsという製品について説明しているまとまった量のサンプル（正事象）と、Stripe という単語の近くに「payments」という単語が出現しているが、製品の名称としてではなく会計用語として使用されている同規模のサンプル（反事象）からなる、十分な量のサンプルでLLMをファインチューニングすることが考えられます。これによって、LLMは、これら 2 つのケースを区別できるようになる可能性があります。

サマリー:

- メリット - 分類器の精度向上
- デメリット - 少なくとも数百のトレーニングサンプルのセットを維持する必要性
- デメリット - クラウドベンダーの提供するファインチューニングモデルは標準のLLMより高額

エンベディングによる分類

分類の3番目のアプローチでは、エンベディングを使用して、ロジスティック回帰、ランダムフォレスト、サポートベクターマシン(SVM)などの従来の分類モデルをトレーニングします。

このアプローチでは、分類器に与えるラベルごとに大量のエンベディングデータが準備可能なトレーニングデータセットを用います。そして、Scikit-Learn や LightGBM などの標準ライブラリを使用して分類器をトレーニングします。

このアプローチは、エンベディングと大規模なトレーニング データセットに基づいており、テキスト データの複雑な分類問題で非常にうまく機能することが期待されます。

サマリー:

- メリット - 分類器の精度向上 (ただし、必ずしも LLM アプローチよりも優れているとは限らない)
- メリット - 適用されるラベルあたりのコストが非常に低い。ベンダーのLLM は、処理されたトークンによって課金されるため、文書が長大、あるいは分類文書セットの数が非常に大規模、またはその両方である場合、LLM 分類器の実行コストは非常に高くなる可能性があります
- デメリット - トレーニング データセットの構築と維持にかかる非常に多くの時間と労力

LLMキャッシング

LLMのAPIコールにはコストがかかり、低速であることもよくあります。多くのシナリオで（例えば、エンタープライズ検索）質問や検索クエリの大多数が、意味的には同じであるのに、単に同じ質問に対して表現が若干異なっているだ

け、ということが見られます。このような場合に、ベクトル検索を活用することにより、クエリを意味レベルで把握し、その内容に応じてキャッシュから結果を検索することができます。

LLMキャッシュのシンプルな実装には、次の手順が必要です。

1. LLM に送信されたクエリごとに、エンベディングを生成します
2. クエリに対する LLM の応答を保存します
3. 両方のエンベディングをベクターデータストアに保存します
4. LLM を呼び出す前に、ベクトル検索を使用して同様のクエリが以前にすでに実行されキャッシュされていないかを確認し、存在する場合はキャッシュされた結果を返します

この手法により、本番アプリケーションで LLM を実行する際のコストとレイテンシの問題を大幅に削減できます。

ただし、欠点の 1 つとして、キャッシュされた応答に個人情報が保存され、別のユーザーの応答に利用されてしまう、ということが起きないように細心の注意を払う必要があります。

エンベディングによる異常検出

異常（アノマリー）検出とは、特定のデータセット内で予想される動作、または通常の動作から大きく逸脱するパターン、あるいはポイントを特定するプロセスを指します。例えば、典型的なパターンに従わない外れ値の検出や、通常と異なる、あるいは非常に稀なイベントの検出、などがあります。

密度ベースの異常検出にベクトル検索を使用する簡単なプロセスは次のとおりです。

- アプリケーションが管理しているデータのエンベディングデータセットを作成
- アプリケーションへの入力（ドキュメントまたは画像）に対して、そのエンベディングデータを生成
- アプリケーションの既存のデータセットの中から入力されたエンベディングデータに最も近いものを検索
- 入力データと検索されたデータとの間の距離が、ノーマルな範囲からの逸脱を示す、しきい値を超えているかどうかを確認
- 入力データを異常（アノマリー）としてラベル付け

参考文献

1. “Economic potential of generative AI.” McKinsey, 2023 年 6 月 14 日, https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/the-economic-potential-of-generative-ai-the-next-productivity-frontier#key-insights
2. “Microsoft and OpenAI extend partnership - The Official Microsoft Blog.” The Official Microsoft Blog, 2023 年 1 月 23 日, https://blogs.microsoft.com/blog/2023/01/23/microsoftandopenaiextendpartnership/
3. Hawk, Jessica. “Build next-generation, AI-powered applications on Microsoft Azure | Azure Blog.” Microsoft Azure, 2023 年 5 月 23 日, https://azure.microsoft.com/en-us/blog/build-next-generation-ai-powered-applications-on-microsoft-azure/
4. “Anthropic Partners with Google Cloud.” Anthropic, 2023 年 2 月 3 日, https://www.anthropic.com/index/anthropic-partners-with-google-cloud
5. “Google AI PaLM 2 – Google AI.” Google AI, https://ai.google/discover/palm2/
6. Try Bard, an AI experiment by Google, https://bard.google.com/
7. “Open LLM Leaderboard - a Hugging Face Space by HuggingFaceH4.” Hugging Face, https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard
8. Colmer, Paul. “Get started with generative AI on AWS using Amazon SageMaker JumpStart | Amazon Web Services.” Amazon AWS, 2023 年 5 月 4 日, https://aws.amazon.com/blogs/machine-learning/get-started-with-generative-ai-on-aws-using-amazon-sagemaker-jumpstart/
9. “Generative AI.” Google Cloud, https://cloud.google.com/ai/generative-ai
10. Fradin, Michelle, and Lauren Reeder. “The New Language Model Stack.” Sequoia Capital, 2023 年 6 月 14 日, https://www.sequoiacap.com/article/llm-stack-perspective/
11. https://langchain-langchain.vercel.app/docs/get_started/introduction.html
12. https://gpt-index.readthedocs.io/en/latest/
13. Wodecki, Ben. “UBS: ChatGPT is the Fastest Growing App of All Time.” AI Business, https://aibusiness.com/nlp/ubs-chatgpt-is-the-fastest-growing-app-of-all-time
14. https://roihacks.com/
15. “Survey reveals AI’s impact on the developer experience.” The GitHub Blog, 2023 年 6 月 13 日, https://github.blog/2023-06-13-survey-reveals-ais-impact-on-the-developer-experience/
16. “90% of VC-Backed Companies Plan to Launch Generative AI in their Products, 64% this Year.” Productboard, 2023 年 6 月 6 日, https://www.productboard.com/blog/generative-ai-and-products/
17. “KPMG Generative AI Survey.” KPMG U.S., https://info.kpmg.us/news-perspectives/technology-innovation/kpmg-generative-ai-2023.html
18. “Most Searched Thing on Google: Top Google Searches in 2023.” Semrush, 2023 年 6 月 13 日, https://www.semrush.com/blog/most-searched-keywords-google/
19. “AutoML Vision documentation.” Google Cloud, https://cloud.google.com/vision/automl/docs

20. Karpathy, Andrej. “Andrej Karpathy on Twitter: "The hottest new programming language is English."” Twitter, 2023年1月24日, https://twitter.com/karpathy/status/1617979122625712128
21. “MTEB Leaderboard - a Hugging Face Space by mteb.” Hugging Face, https://huggingface.co/spaces/mteb/leaderboard
22. https://platform.openai.com/docs/guides/embeddings/types-of-embedding-models
23. Instructor Text Embedding, https://instructor-embedding.github.io/
24. “Libraries - OpenAI API.” Platform OpenAI, https://platform.openai.com/docs/libraries
25. “Vertex AI.” Google Cloud, https://cloud.google.com/vertex-ai