Azure OpenAI Serviceによる

RAG実装ガイド

～ 生成AIアプリケーションの解説と実践 ～

はじめに

本書の目的

本書の目的は、「シンプル」「強力」「すぐ動く」をモットーにした RAG アプリケーションを実装するためのガイドであり、これらか RAG を始める人に参考にしてもらうべく一筆したためました。本書では RAG のアーキテクチャのみならず「実際に動くコード」もご用意致しました。読者の皆様には、コードを動かしながら RAG をより深くご理解頂けることを一番の目的としております。

RAG、つまり Retrieval-Augmented Generation は、とても便利ですが、一見してその全貌を掴むのは少々難しいものがあります。

そこで、このガイドでは、初心者の方々でもスムーズに RAG の世界に入っていただけるよう、分かりやすいサンプルコードとその解説を用意しました。これを通じて、RAG を誰よりも深く理解し、使いこなせるようになることが私の願いです。

再び強調しますが、このガイドで紹介するアプリケーションは、「シンプル」「強力」「すぐ動く」を理念としています。

- **シンプル**
  余計な機能がなく、シンプルなチャットのインターフェースのみを提供しています。これにより、ソースコードの可読性が高まり、中身を理解しやすくなっています。
- **強力**
  シンプルとはいえど機能は強力です。Azure の最先端の検索手法である「セマンティックハイブリッド検索」、Azure Static Web Apps、Azure Functions によるフルサーバーレス構成といった最新技術が集結されています。
- **すぐ動く**
  Azure Developer CLI、Bicep によって IaC(Infrastructure as Code) を実現しており、コマンド一発で Azure にリソースがデプロイされて動くようになっています。

本書で提供する RAG アプリケーションは以下の Git リポジトリで公開しております。

https://github.com/noriyukitakei/aoai-rag-starter

本書の構成

本書の構成は以下のようになっております。

第 1 章「RAG の概要」

RAG とはどういうものか、なんの役に立つのかというのを詳細に解説しております。

第 2 章「Azure AI Search」

RAG を構成する重要なパーツの一部である外部データベースとしての Azure AI Search について解説しております。ベクトル検索やハイブリッド検索、セマンティック検索の成り立ちを説明しており、少々ボリュームのある章ではありますが、今回本書でご提供する RAG アプリケーションをご理解頂くには欠かせない内容となっております。

第 3 章「RAG アーキテクチャ」

本書で提供する RAG アプリケーションのシステム構成や画面構成といった内容を解説しております。

第 4 章「ビルド・デプロイの仕組み」

本書で提供する RAG アプリケーションはビルド・デプロイが IaC によって自動化されており、その構成や手順を解説しております。

第 5 章「リポジトリの構成」

本書で提供する RAG アプリケーションのファイルやディレクトリ構成、そしてそれらの中でも主要なファイルの役割を解説しております。

第 6 章「Azure へのデプロイ」

本書で提供する RAG アプリケーションを Azure で動作させるための方法が記載されております。まずはサクッと動かしてみたい方は、いきなり本章をお読み頂き、手順を実施しても OK です。

第 7 章「開発環境」

本書で提供する RAG アプリケーションを開発環境で動作し、デバッグするための方法を記載しております。

本書の対象読者

本書では次のような人を対象としています。

- RAG に興味はあるけど、実装したことない人
- Azure 上で RAG を実装してみたい人

前提とする知識

本書を読むにあたり、次のような知識が必要となります。

- Azure についての基礎知識
- Python による簡単な Rest API の開発経験
- React による簡単な SPA の開発経験
- Azure Open AI Service の基本的な操作

問い合わせ先

本書に関する質問やお問い合わせは、以下までお願いいたします。

- X: @noriyukitakei
- mail: mail@noriyukitakei.jp

謝辞

本書は、マイクロソフト社から提供されている「Azure OpenAI Service リファレンスアーキテクチャ」を参考に致しました。

https://www.microsoft.com/ja-jp/events/azurebase/contents/default.aspx?pg=AzureOAIS

このような素晴らしいドキュメントを提供してくださったマイクロソフトの方々には海より深く感謝申し上げます。

目次

はじめに i

第 1 章 RAG の概要 1
1.1 生成 AI によるチャットシステム 1
1.2 独自情報に基づいたチャットシステム 2
1.3 RAG とは？ 3
1.4 RAG のない世界と RAG のある世界 4
1.4.1 RAG のない世界 4
1.4.2 RAG のある世界 4
1.4.3 RAG のない世界と RAG のある世界の違い 5

第 2 章 Azure AI Search 7
2.1 外部データベースの役割 7
2.2 Azure AI Search 8
2.3 キーワード検索 9
2.3.1 転置インデックス 9
2.3.2 検索結果の順位付け 11
2.3.3 「猫」という単語で検索した場合 12
2.3.4 「素晴らしい」という単語で検索した場合 13
2.4 ベクトル検索 17
2.4.1 ベクトル検索をわかりやすい事例で考える 17
2.4.2 ベクトル検索を試してみる 20
2.5 セマンティック検索 28
2.6 ハイブリッド検索 40
2.7 セマンティックハイブリッド検索 46

第 3 章 RAG アーキテクチャ 49
3.1 システム構成 49
[コラム] Document Intelligence とは？ 53
[コラム] Azure Static Web Apps とは？ 54
[コラム] Azure Functions とは？ 55
[コラム] Azure Cosmos DB とは？ 56
3.2 画面の構成 57

3.3 フロントエンドとバックエンドの連携 58
3.4 インデクサー 61
3.4.1 チャンク化 61
3.4.2 オーバーラップ 61
3.4.3 インデックスまでの流れ 63

第 4 章 ビルド・デプロイの仕組み 65
4.1 インフラの自動化と IaC 65
4.2 IaC を実現するツール 66
4.2.1 Bicep 66
4.2.2 Azure Developer CLI 66
4.3 ビルドとデプロイの流れ 68

第 5 章 リポジトリの構成 71
5.1 リポジトリの構成 71
5.2 data ディレクトリ 73
5.3 scripts ディレクトリ 74
5.3.1 scripts/postprovision.sh 74
5.3.2 scripts/postprovision.ps1 74
5.3.3 scripts/indexer.py 74
5.3.4 scripts/cosmosreadwriterole.json 75
5.4 infra ディレクトリ 76
5.4.1 infra/main.bicep 76
5.4.2 infra/main.parameters.json 76
5.4.3 infra/modules 76
5.5 src ディレクトリ 77
5.5.1 src/frontend 77
5.5.2 src/frontend/src/App.tsx 77
5.5.3 src/frontend/src/components/ ChatQuestion/ChatQuestion.tsx 77
5.5.4 src/frontend/src/components/ ChatAnswer/ChatAnswer.tsx 77
5.5.5 src/frontend/src/components/ InputQuestion/InputQuestion.tsx 77
5.5.6 src/frontend 78
5.5.7 src/backend/function_app.py 78
5.5.8 src/backend/local.settings.json 79

第 6 章　Azure へのデプロイ　81
6.1　Azure にデプロイされるリソース 81
6.2　事前準備 82
6.3　デプロイ手順 83
6.4　動作確認 85

第 7 章　開発環境　87
7.1　事前準備 87
7.2　開発環境の構成 88
7.3　デバッグ方法 90
7.3.1　Azure Functions Core Tools の起動 91
7.3.2　Node.js の起動 92
[コラム] Azure Static Web Apps CLI とは？ 93
[コラム] Azure Functions Core Tools とは？ 94

第 1 章

RAG の概要

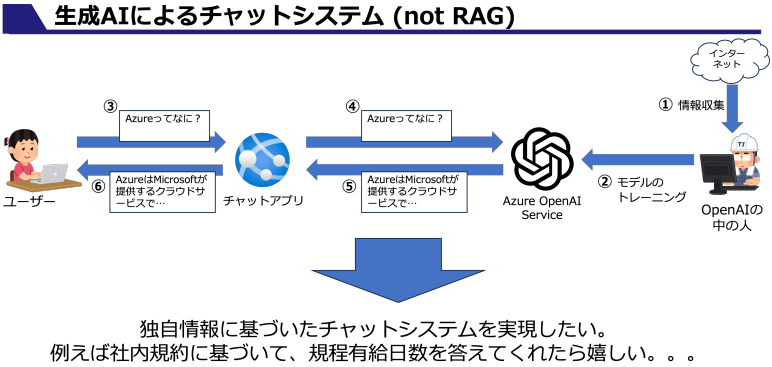
本章では、RAG(Retrieval-Augmented Generation) の概要を説明します。

1.1 生成 AI によるチャットシステム

「AI がお答えします！」こんなフレーズを目にすると、我々はすぐに高度なテクノロジーを思い浮かべるかもしれません。例えば、ChatGPT のようなシステムは、豊富な一般知識をもとに会話を行いますが、その知識は、インターネットから収集されたデータに基づきあらかじめトレーニングされたモデルに基づいています。このシステムが得意とするのは、広範囲にわたるトピックへの即興の応答です。しかし、ここには限界があります。最新の出来事や特定の企業の内部情報のような、トレーニングデータにない知識を質問された時、この AI は手をこまねいてしまうのです。

1.2　独自情報に基づいたチャットシステム

では、特定の分野の専門家として AI を働かせるにはどうすればいいでしょうか。一つの答えは、「モデルのトレーニング」です (図 1.1 参照)。大量のドメイン特化データを集め、時間をかけ、費用を投じて AI を教育するのです。しかしこの方法は、新しい情報が次から次へと生まれるこの時代に追いつくのが難しい上、コストもかかり、さらには必要なデータが手に入らないこともしばしばです。このような状況は、特にスタートアップや中小企業にとっては大きな壁となります。



▲図 1.1: 独自情報に基づいたチャットシステム

1.3　RAG とは？

ここで「RAG」、つまり Retrieval-Augmented Generation の登場です。これは革新的なアプローチで、トレーニング済みの AI に新たな知識を"検索"させて回答を導く技術です。まるで図書館で最適な資料を探し出すリサーチャーのように、RAG は AI に外部からの情報を取り入れさせ、その場で必要な知識を引き出します。これにより、AI は学習したことのない最新のトピックにも対応可能となり、知識の範囲を大幅に拡張します。RAG を使えば、企業は自社の独自情報を AI に活用させることができるため、より個別化され、詳細な対話が可能となります。それはまるで、吉野家の牛丼が「速い、安い、うまい」という価値を提供するように、RAG は「簡単に、強力に、すぐに」知識を提供するのです。

例えば、就業規則のような独自データに基づいた回答を生成 AI がしてくれたら、とっても便利だというのは言うまでもありません。今まで一所懸命就業規則を検索して調べてたのが、生成 AI に「育児休業の申請方法ってどうすればいいの？」と聞くだけで、回答が返ってくるなんて、とってもステキだと思いませんか？

それを実現してくれるのが RAG(Retrieval-Augmented Generation) です。

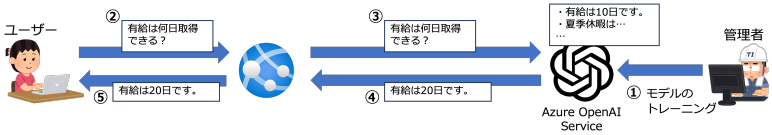
1.4　RAG のない世界と RAG のある世界

RAG のような新しい技術は、それがない世界とある世界を比較して、何がどう良くなるのかを

ということで、まずは RAG のない世界を見てみます。

♣ 1.4.1　RAG のない世界

RAG のない世界、つまり RAG を使わずに独自情報に基づいたチャットシステムを実現するには「モデルのトレーニング」が必要であることを先程解説しました。



▲図 1.2: RAG のない世界

図 1.2 を見てください。RAG のない世界では、ユーザーからの質問に Azure OpenAI Service は以下のような手順で回答を返します。

① 管理者はモデルのトレーニングを行います。これは、質問とそれに対して期待される回答のペアを Azure OpenAI Service に登録します。精度の高い回答を出力するためには、可能な限り多くの高品質なデータを使用することが望ましいです。

② ユーザーは、チャットの UI を提供しているアプリケーションに対して「有給は何日取得できる？」と入力します。

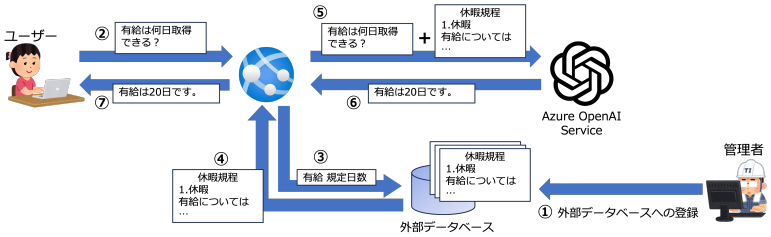
③ アプリケーションは、Azure OpenAI Service に対して、先程のユーザーの質問を投げます。

④ Azure OpenAI Service は、トレーニングされたモデルに基づいて回答をアプリケーションに対して返却します。

⑤ アプリケーションは、Azure OpenAI Service から受け取った回答をユーザーに返却します。

♣ 1.4.2　RAG のある世界

では、次に RAG のある世界を見てみます。



▲図 1.3: RAG のある世界

図 1.3 を見てください。RAG のある世界では、ユーザーからの質問に Azure OpenAI Service は以下のような手順で回答を返します。

① 管理者は外部データベース (Azure AI Search などの全文検索システムなど) に、回答のための情報源を登録する。例えば、社内規約が記載された PDF ファイル等である。
② ユーザーは、チャットの UI を提供しているアプリケーションに対して「有給は何日取得できる？」と入力します。
③ アプリケーションは、ユーザーの質問をもとに、外部データベースに対して、回答に必要な情報を検索します。
④ 外部データベースは、アプリケーションに対して検索結果を返します。
⑤ アプリケーションは、④で取得した情報に基づいてユーザーからの質問に回答するよう Azure Open Service に依頼します。
⑥ Azure OpenAI Service は、アプリケーションに回答を返します。
⑦ アプリケーションは、ユーザーに回答を返します。

♣ 1.4.3　RAG のない世界と RAG のある世界の違い

「RAG のない世界」と「RAG のある世界」も、ユーザーから見た挙動に変わりはありません。その違いは管理者の手間にあります。「RAG のない世界」では、管理者が Azure OpenAI Service に対して「モデルのトレーニング」を実施しています。これは一般的には「モデルの微調整」とも言われる作業で、これはかなり骨の折れる作業です。モデルの微調整は、学習済みのモデルに追加の情報やデータを組み込むことで、その性能や反応を調整するプロセスなのですが、新しいデータセットの用意、学習の設定やパラメータ調整、そして再学習の実行など、多くの手間と時間がかかります。

例えば、就業規則の例で言えば「有給休暇は何日取得できる？」というユー

ザーからの質問に「10 日です」と回答させるためには以下のデータセットを作成して、Azure OpenAI Service に登録する必要があります。

```
{"prompt": "有給は何日取れますか？", "completion": "10日です。"}
```

就業規則に何でも答えるチャットシステムを作るためには、就業規則に基づき上記のようなデータをたくさん作らなければいけません。これは相当骨の折れる作業になるというのは想像に難くないと思います。

マイクロソフトも「モデルの微調整は最後の手段」と謳っています。

一方で「RAG のある世界」ではどうでしょうか？ この場合、管理者は社内に既に存在する情報源を外部データベースに登録するだけで済みます。例えば、就業規則に関する質問に答えるチャットシステムを作る場合、社内にある就業規則の PDF ファイルを外部データベースに登録するだけです。その後、アプリケーションはユーザーの質問に関連する情報を外部データベースから取得し、その情報をもとに Azure OpenAI Service に回答の生成を依頼します。

RAG は管理者の負担を削減しつつ、ユーザーに対して適切な回答を提供するための効率的な手段であると言えます。

第 2 章

Azure AI Search

この章では、RAG の重要な構成要素である「外部データベース」としてよく利用される Azure AI Search について説明します。

2.1 外部データベースの役割

第 1 章「RAG の概要」の「♣ 1.4.2 RAG のある世界」(p.4) で説明したように、RAG では外部データベースが重要な役割を果たします。このデータベースは、生成モデルが回答を生成する際に参照する情報源として機能します。ユーザーからの質問に対して、外部データベースから関連する情報を検索し、その情報をもとに回答が生成されます。このプロセスにより、生成モデルはより正確で詳細な回答を提供することができます。

つまり、ユーザーからの質問に正確に回答するには、外部データベースの検索精度が重要というわけです。

2.2 Azure AI Search

Azure AI Search は、外部データベースとして RAG でよく利用されるサービスです。これは、大量のデータから関連する情報を迅速に検索できる高度な検索機能を提供します。Azure AI Search を使用することで、RAG モデルは質問に対する回答を生成する際に、必要な情報を効率的に取得できます。

Azure AI Search は、自然言語処理技術を活用して、検索クエリの意図を理解し、関連性の高い結果を返すことができます。これにより、ユーザーの質問に対してより適切な情報を提供することが可能となります。また、Azure AI Search はスケーラブルであり、大規模なデータセットに対しても高速な検索性能を維持することができます。これは、RAG モデルを企業レベルで運用する際に特に重要な特徴です。

そして、Azure AI Search は「ベクトル検索」という機能を提供しています。ベクトル検索は、文書やクエリを高次元のベクトル空間にマッピングし、その空間内での類似度に基づいて検索を行う技術です。この方法により、意味的な類似性に基づいて情報を検索することができるため、単語の表層的な一致に頼る従来の検索手法よりも精度の高い結果を得ることが可能です。

Azure AI Search のベクトル検索機能を活用することで、RAG モデルはユーザーの質問に対する回答を生成する際に、より関連性の高い情報を効率的に取得できます。これにより、生成される回答の質が向上し、ユーザーに対してより適切な情報を提供することができるようになります。ベクトル検索は RAG の性能を大幅に向上させるため、現在の RAG においては非常に重要な技術となっています。

さらに、Azure AI Search では、このベクトル検索の精度をより高めるために「セマンティック検索」「ハイブリッド検索」「セマンティックハイブリッド検索」があります。そして、本ガイドで提供するアプリケーションは、現在最も検索精度が高いと言われている「セマンティックハイブリッド検索」を利用しています。

少々迂遠ではありますが、本章では「セマンティックハイブリッド検索」を説明するために、従来から最もよく利用されている「キーワード検索」、先ほど説明した「ベクトル検索」、マイクロソフトの独自モデルをベースとした「セマンティック検索」、ベクトル検索とキーワード検索を組み合わせた「ハイブリッド検索」、そして最後にハイブリッド検索とセマンティック検索を組み合わせた「セマンティックハイブリッド検索」の順に説明します。

2.3　キーワード検索

これは従来から使われている検索方法です。転置インデックスというものを作成し、その転置インデックスから検索をして、対象のドキュメントを引っ張ってくる方法です。

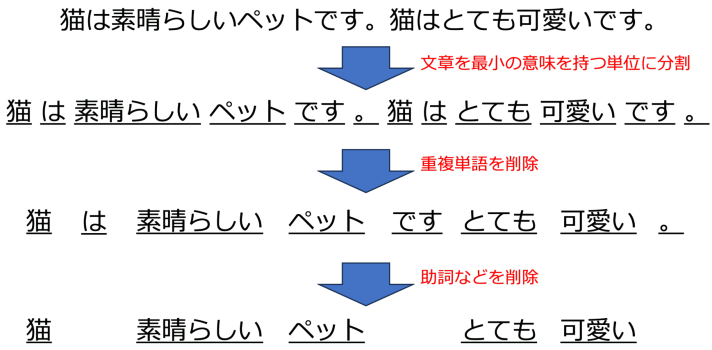
♣ 2.3.1　転置インデックス

例えば、以下の 4 つのドキュメントを考えてみます。

- ドキュメント 1: 猫は素晴らしいペットです。猫はとても可愛いです。
- ドキュメント 2: 犬は素晴らしいペットです。犬は忠実です。
- ドキュメント 3: 鳥は素晴らしい歌を歌う。鳥は美しいです。
- ドキュメント 4: 猫と犬は人気のペットです。

まずはこれを形態素解析します。形態素解析とは、文章を最小の意味を持つ単位（形態素）に分割し、それぞれの品詞を特定するプロセスです。例えば、日本語の文章では、単語を切り分けて名詞や動詞などに分類します。これは、テキストデータを解析する際の基本的な手順であり、自然言語処理の分野で広く用いられています。

これに従い、先程の文章を形態素解析するプロセスは図 2.1 の通りとなります。



▲図 2.1: 形態素解析のプロセス

同様に他のドキュメントも形態素解析を行い、その結果を以下に記載します。

- ドキュメント 1: 猫　素晴らしい ペット　猫　とても 可愛い
- ドキュメント 2: 犬　素晴らしい ペット　忠実
- ドキュメント 3: 鳥　素晴らしい 歌　歌う　鳥　美しい
- ドキュメント 4: 猫　犬　人気　ペット

形態素解析した結果をもとに転置インデックスを表 2.1 のように作成します。転置インデックスは、文書検索システムで使用されるデータ構造の一つで、各単語がどの文書に現れるかを記録するためのものです。例えば、今回の例だと、「猫」という単語はドキュメント 1 とドキュメント 4 に出現しますので、そのように記録します。そしてのこの転置インデックスに対して、「猫」という単語で検索すると、関連ドキュメントであるドキュメント 1 とドキュメント 2 を素早く見つけることができます。

SQL の like 文ではだめなのかという疑問が湧くと思います。like 文検索では全ての文書を検索しなければなりません。例えば、SQL で「ペット」で like 文検索 (select * from docs where content like ‘% ペット %’) したとします。そうなると、文書全体を検索することになります。

一方で、転置インデックスを検索する場合は、「ペット」をキーに検索しますと、その単語に直接関連付けられた文書 (ドキュメント 1、ドキュメント 2、ドキュメント 4) を引っ張ってくることができます。これにより、検索時間が大幅に短縮され、大量のデータを扱う検索エンジンで特に有効です。

続きをご覧になりたい方は
以下のURLから
ダウンロードお願い致します

https://mk.sios.jp/aoai-rag-dev-guide-form