

Lesson 23 【最終課題】各ユースケースごとのLLMアプリ開発

Chapter 2 サンプルアプリ①「社内情報特化型生成AI検索アプリ」

学習時間の目安 20時間

当Chapterでは、社内文書をもとにユーザーからの入力に対して回答する生成AIチャットボットを題材とし、本格的なLLMアプリのコード解読、また機能改修・追加に取り組んでもらいます。

当Chapterの目的

ここまで学習を通じて、本格的なLLMアプリを開発するための基礎的なスキルは習得済みです。習得したスキルを活用し、当Chapterでは初めての本格的なLLMアプリのコードに触れてもらいます。

当Chapterで紹介するLLMアプリは、これまでと比較にならない量のコードです。これまでに学んだことをベースとしたコードであるため、多少の調査は必要となりますが自力でコードを解読できるものです。しかしコード解読に加え、提出課題での機能改修・追加には苦労するかと思います。

当Chapterの目的は、以下の3点です。

1. 本格的なLLMアプリの作り方を理解する。
2. 長くて複雑なコードを解読し、試行錯誤しながら機能改修・追加ができるようにする。
3. ゼロベースからでも本格的なLLMアプリを開発できるようにする。

当Chapterで紹介するLLMアプリのコードを解読し、コードの書き換えができるようになれば、次のChapter以降で紹介する3つのLLMアプリについてもスムーズにコードの意味を理解し、書き換えができるようになります。

次からの3つのChapterよりも多くの提出課題を用意していますので、まずは当Chapterでしっかり時間をかけてコード解読（インプット）、機能改修・追加

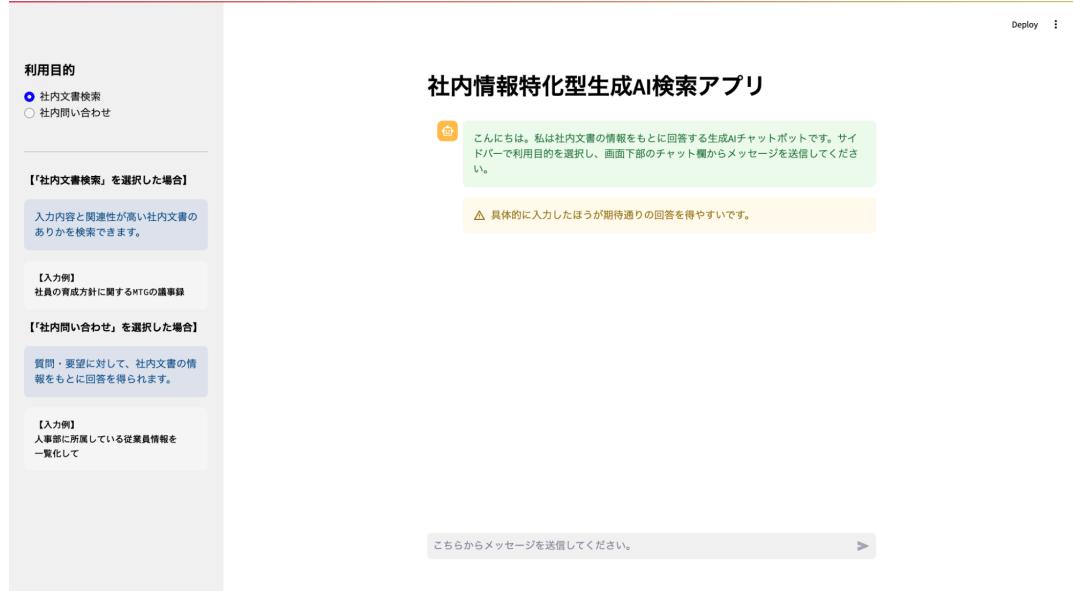
(アウトプット) に取り組んでもらえればと思います。

アプリの概要

当Chapterでは、「社内情報特化型の生成AI検索アプリ」を題材として扱います。

以下は、提出課題に取り組んでもらった後の完成版のアプリ画面です。

(提出課題に取り組む前の段階では、サイドバーを使っていなかったり、内部の挙動に一部不具合が発生していたりする状態です)



機能は、大きく「社内文書検索」と「社内問い合わせ」の2つに分かれており、サイドバーから利用目的を選択できます。

社内文書検索

まず「社内文書検索」では、ユーザーがチャット欄から送信した入力をもとに社内文書を検索し、入力内容と関連性が高い社内文書のありかが表示されます。

例えば「社員の育成方針に関するMTGの議事録」と入力すると、以下のように社内文書のありかが表示されます。

利用目的

- 社内文書検索
- 社内問い合わせ

【「社内文書検索」を選択した場合】

△ 具体的に入力したほうが期待通りの回答を得やすいです。

【入力例】
社員の育成方針に関するMTGの議事録

【「社内問い合わせ」を選択した場合】

△ 入力内容に関する情報は、以下のファイルに含まれている可能性があります。

① ./data/MTG議事録/教育/教育ミーティング議事録.docx
② ./data/MTG議事録/全社/全社ミーティング議事録.docx

その他、ファイルあらかじめ候補を提示します。

③ ./data/MTG議事録/採用/採用ミーティング議事録.docx

こちらからメッセージを送信してください。

どこの会社でも、社内には大量の資料が存在するものです。「この資料、どこにあったかな」と資料探しに多くの時間をかけている人も多いですが、生成AIを活用した社内文書検索のアプリを作れるようになれば、各従業員の資料探しにかかる工数を削減でき、全社的な生産性向上・コスト削減を実現できます。

社内問い合わせ

次に「社内問い合わせ」では、ユーザーがチャット欄から送信した入力と関連性が高い社内文書を検索し、ヒットした社内文書をもとに回答が返されます。顧客問い合わせの自動化と並び、社内問い合わせの自動化は生成AIチャットボットが特に効果を発揮するところです。

「人事部に所属している従業員情報を一覧化して」と入力すると、以下のように社内文書をもとに回答が表示されます。

利用目的

- 社内文書検索
- 社内問い合わせ

△ 具体的に入力したほうが期待通りの回答を得やすいです。

【「社内文書検索」を選択した場合】

△ 人事部に所属している従業員情報を一覧化して

【入力例】
人事部に所属している従業員情報を一覧化して

【「社内問い合わせ」を選択した場合】

△ 人事部に所属している従業員情報を一覧化して

従業員区分	入社日	役職	スキルセット	保有資格	大学名	学部・学科
インター ン	2017-06-08	アシスタント	営業スキル、Web開発、データベース設計、クラウドコンピューティング	PMP（プロジェクトマネジメント）、秘書検定2級	慶應義塾大学	理学部
契約社員	2022-07-14	インター ン	人事管理、Excel、プログラミング開発、Python	簿記2級、ITパスポート	千葉大学	文学部

こちらからメッセージを送信してください。

また回答だけでなく、以下のように参照した社内文書のありかも表示されます。

部署	大学部	情報セキュリティマネジメント	AWS認定ソリューションアーキテクト、Salesforce認定アドミニストレーター	スタッフ	2021-05-22	インター	EMP0047	性別	年齢	登録日	担当者名	メールアドレス	会員登録	性別	年齢	登録日	担当者名	メールアドレス	会員登録
経済学部	筑波大学	情報セキュリティマネジメント	AWS認定ソリューションアーキテクト、Salesforce認定アドミニストレーター	スタッフ	2021-05-22	インター	EMP0047	女性	34歳	1999-07-16	26	yumiiko77@example.com	会員登録	女性	34歳	10-27	49	kenichi45@example.com	会員登録

この表は人事部に所属している従業員の情報を一覧化したものです。

情報源

- ./data/社員について/社員名簿.csv
- ./data/MTG議事録/採用/採用ミーティング議事録.docx
- ./data/MTG議事録/採用/採用.pdf (ページNo.1)

こちらからメッセージを送信してください。▶

社内問い合わせ対応自動化の生成AIチャットボットを開発できるようになれば、ヘルプデスク宛に各従業員から何度も同じ質問が寄せられて対応に工数を要しているといった悩みを解消できます。

また具体的な活用場面の一例として、新入社員によくある「分からないことがあるが、誰に聞けばいいか分からない」「何度も質問するのは気が引ける」といった悩みを解消できます。

社内文書との関連性が低い入力に対しての回答

社内文書との関連性が低い入力が与えられた場合、「社内文書検索」が選択されている状態であればLLMは「入力内容と関連する社内文書が見つかりませんでした」、「社内問い合わせ」が選択されている状態であれば「回答に必要な情報が見つかりませんでした」と回答します。

【「社内文書検索」が選択されている場合】



今日の東京の天気は？



入力内容と関連する社内文書が見つかりませんでした。

入力内容を変更してください。

【「社内問い合わせ」が選択されている場合】



好きな食べ物は？



回答に必要な情報が見つかりませんでした。

まさに、RAGを活用した本格的なLLMアプリであることが分かるかと思います。

ソースコードのありか(3/31までに受講開始された方向け)

Google ドライブの当Lesson用の受講者フォルダ内に、以下のフォルダが置かれているためダウンロードしましょう。

【フォルダのありか】

「Lesson23: 【最終課題】各ユースケースごとのLLMアプリ開発」フォルダ

「Chapter2: サンプルアプリ①「社内情報特化型生成AI検索アプリ」」フォルダ

「ダウンロード用」フォルダ

「company_inner_search_app」フォルダ

ソースコードの取得(4/14以降に受講開始された方向け)

下記のリンクを押下し、ソースコードのフォルダをダウンロードしましょう。

Chapter2: サンプルアプリ①「社内情報特化型生成AI検索アプリ」

中身には「ダウンロード用」と「提出用」のフォルダがあり、前者に
「company_inner_search_app」フォルダが格納されています。

ステップ1：ダウンロードしたZipファイルの解凍

ダウンロードしたファイルは、そのままでは使えません。「解凍」という作業で、中のファイルを取り出す必要があります。

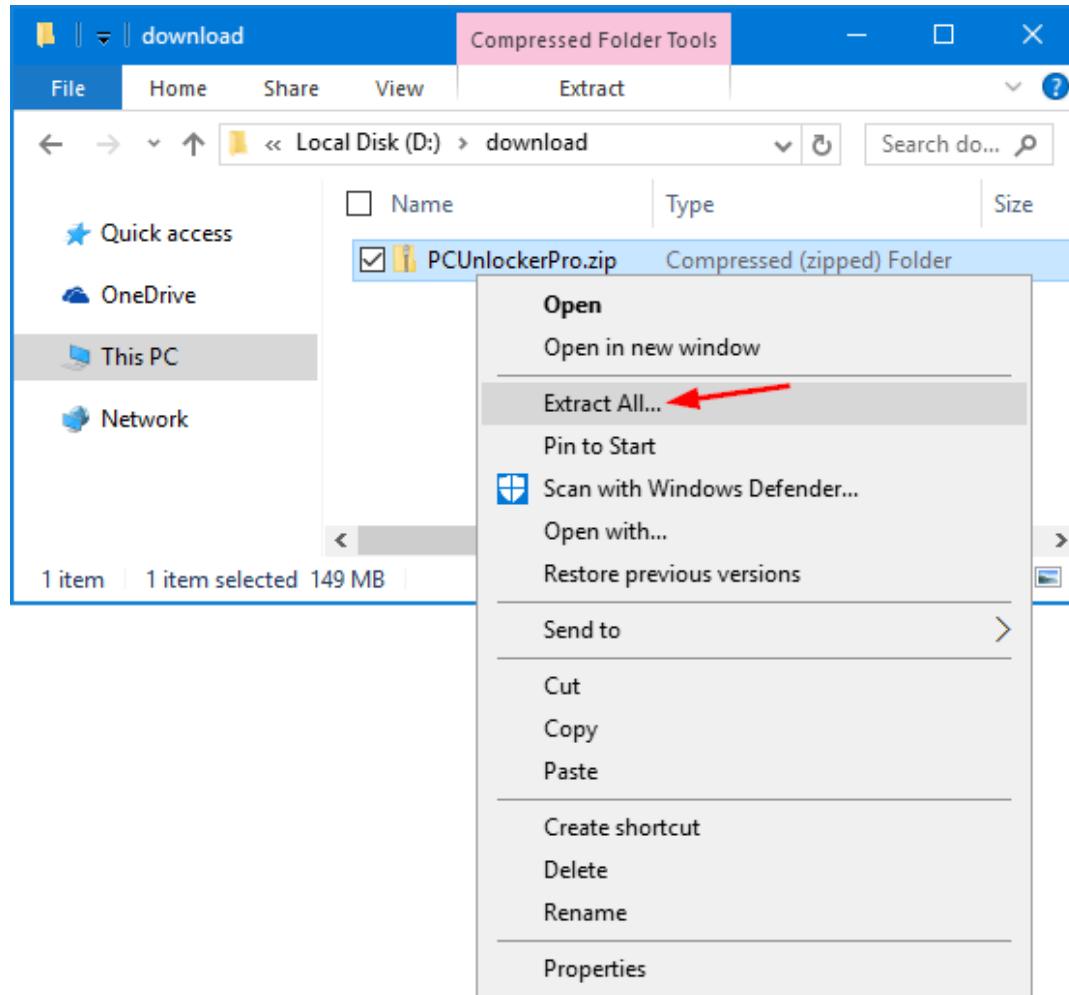
1.ダウンロードフォルダを開く：

- 通常、ダウンロードしたファイルはPCの「ダウンロード」フォルダに保存されています。エクスプローラー (Windows) やFinder

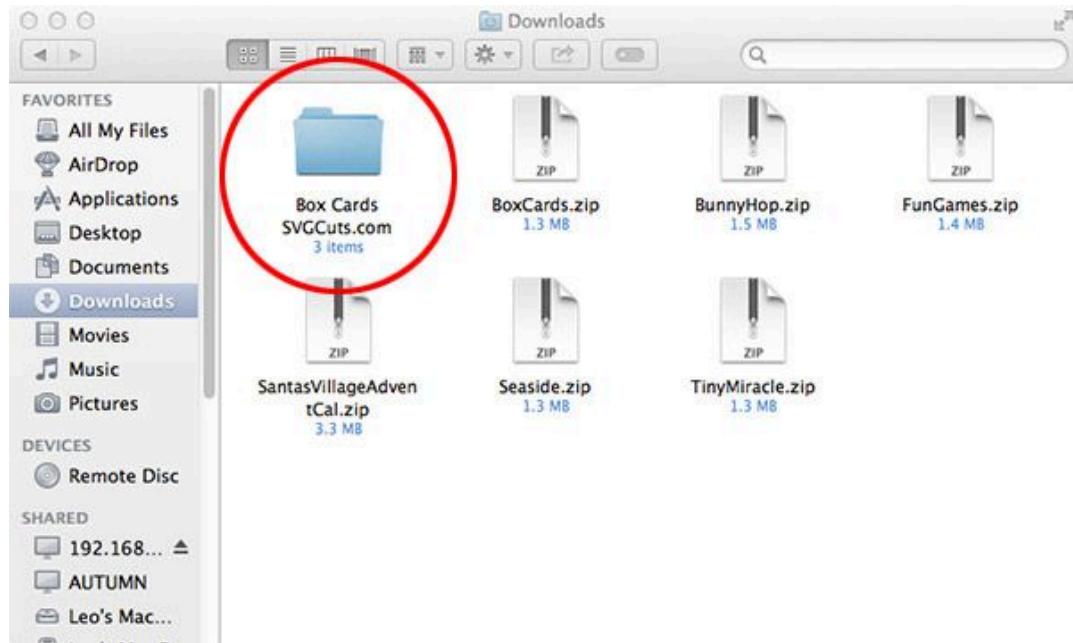
(Mac) で「ダウンロード」フォルダを開いてください。

2.Zipファイルを解凍する:

- **Windowsの場合:** 1.ダウンロードしたzipファイルを右クリックします。 2.メニューから「すべて展開」または「Extract All」を選択します。 3.展開先のフォルダを確認する画面が表示されるので、特に変更がなければ「展開」ボタンをクリックします。 4.同じ名前の新しいフォルダが作成され、その中にファイルが展開されます。



- **macOSの場合:** 1. ダウンロードしたzipファイルをダブルクリックします。 2. 自動的に解凍が始まり、同じ名前の新しいフォルダが作成され、その中にファイルが展開されます。



3.解凍されたフォルダを確認する:

- 解凍後、zipファイルと同じ名前の通常のフォルダができていることを確認してください。このフォルダの中に、教材や提出課題のフォルダが入っています。

ステップ2：Google Driveを開く

次に、課題を提出するためのあなた専用のGoogle Drive フォルダを開きます。

※Slackの「生成ai_〇〇_エンジニア_〇〇様」の個人チャンネル、関連ページに添付している「学習フォルダ」を指します。

ステップ3：解凍したフォルダをGoogle Driveにアップロードする

1. Google Driveの画面を確認:

ステップ2で開いた、受講生様個人専用のGoogle Drive フォルダが表示されていることを確認します。

2. 「+ 新規」ボタンをクリック: 画面左上にあるカラフルな「+ 新規」ボタンをクリックします。



3. 「フォルダのアップロード」を選択: 表示されたメニューの中から「フォルダのアップロード」を選択します。



4. アップロードするフォルダを選択:

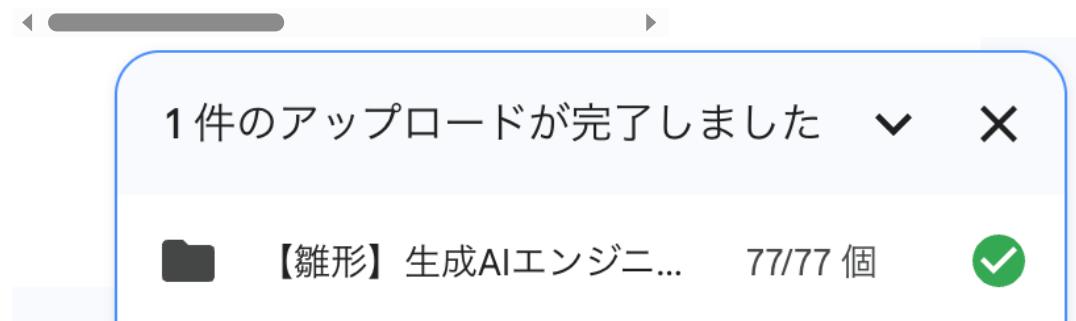
- ファイル選択のウィンドウが開きます。ここで、**ステップ1で解凍したフォルダを選択してください。**
- フォルダを選択したら、「アップロード」または「開く」ボタンをクリックします。

5. アップロードの確認:

- 確認のメッセージが表示されたら、「アップロード」をクリックします。



- 画面右下にアップロードの進捗が表示されます
- アップロードが完了すると、Google Drive



事前準備

Windowsの場合のみ、ソフトウェア「Microsoft Visual C++」のバージョン14.0以上のインストールが必要です。インストールされていない場合、以下の手順に沿って対応してください。

Macの場合はWindows / Mac 共通の手順のみ対応してください。

Windowsの場合のみ

まずは以下のリンクにアクセスし、「Microsoft C++ Build Tools」のダウンロードページを開きましょう。

<https://visualstudio.microsoft.com/ja/visual-cpp-build-tools/>

すると以下の画面が表示されます。



Microsoft C++ Build Tools は、Visual Studio を使用しない、スクリプトで実行可能なスタンドアロン インストーラーを通じて MSVC ツールセットを提供します。Windows を対象とする C++ ライブラリとアプリケーションをコマンドラインからビルドする場合 (例: 構築のインテグレーション ワークフローの一部として)。Visual Studio 2015 Update 3, Visual Studio 2017, Visual Studio 2019、および Visual Studio 2022 の最新版に同梱されたツールが含まれています。

[チュートリアル: コマンド ラインでのネイティブ C++ プログラムのコンパイル](#) →

Build Tools の使用方法

ファイルバック

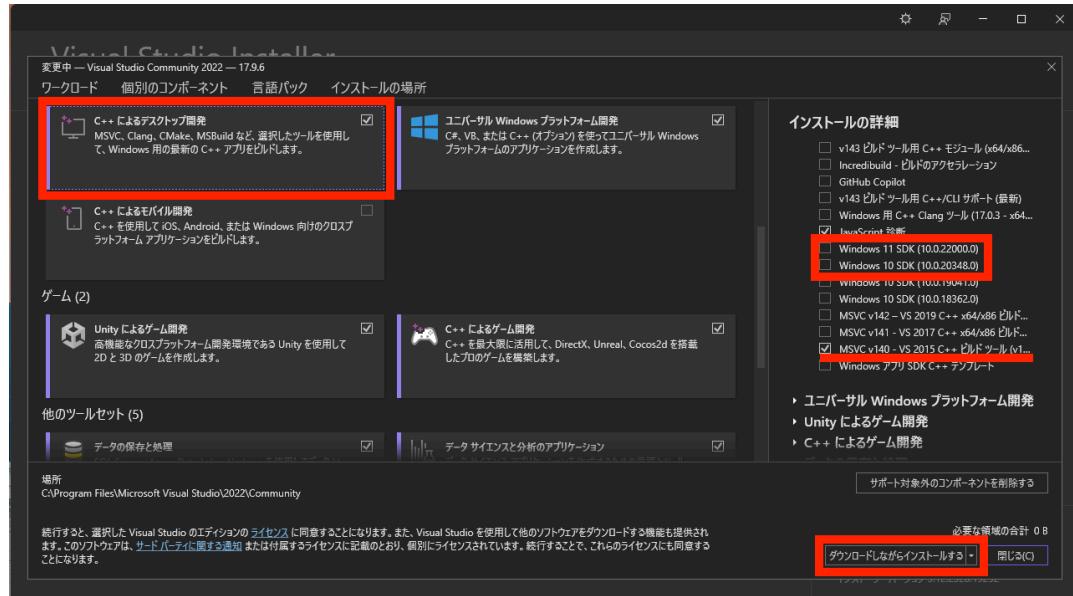
「Build Tools のダウンロード」ボタンをクリックし、インストーラーの実行ファイルをダウンロードしましょう。

ダウンロードされた実行ファイルを起動し、手順に沿って「Microsoft C++ Build Tools」のインストールを進めてください。

インストールが完了すると、ソフトウェアが起動します。起動されない場合はスタートメニューから「Visual Studio Installer」を探し、起動しましょう。

「Visual Studio Build Tools 2019」の「変更(M)」をクリックし、「C++によるデスクトップ開発」にチェックを入れます。

(画面の表示内容は以下とは異なります)



また右側に表示されるインストールの詳細で「MSVC v140」という項目にチェックを入れましょう。さらに、ご自身の環境に合ったWindows SDK（「Windows 11 SDK」or 「Windows 10 SDK」）にもチェックを入れましょう。

チェックを入れたら、右下の「ダウンロードしながらインストールする」、もしくはその右側に「変更」ボタンが表示されている場合は「変更」ボタンをクリックしてインストールを行いましょう。

この後、パッケージのインストールを行った際に「Microsoft Visual C++ 14.0 or greater is required.」というエラー文が表示された場合、PCを再起動して再度パッケージのインストールを試してみてください。

Windows / Mac 共通

1. ダウンロードしたソースコードのフォルダをVSCodeにセットしましょう。
2. Python仮想環境の作成・有効化を行いましょう。
3. フォルダ直下に、ソースコードを正常に動作させるためのパッケージ情報の一覧が記載されている「requirements_windows.txt」「requirements_mac.txt」のファイルが置かれています。Windows の場合は「pip install -r requirements_windows.txt」、Macの場合は「pip install -r requirements_mac.txt」のコマンドを実行し、パッケージの一覧をインストールしましょう。
4. 「Lesson21: Streamlitを活用したWebアプリ開発」の「Chapter5: Streamlit Community Cloudへのデプロイ」を参考に、必要に応じてGitHubと接続しておきましょう。
5. 「.env」ファイルに「OPENAI_API_KEY=ご自身のAPIキー」の形式で、APIキーをセットしましょう。
6. 「streamlit run main.py」のコマンドを実行し、Streamlitアプリを起動しましょう。

今回の題材を通じて新たに学べること

これから、社内情報特化型の生成AI検索アプリについてソースコードの解読に取り組んでもらいます。ソースコードには、これまでのLessonを通じてまだ学んでいない技術が数多く出てきます。

しかし、これまでの学習を通じて「LangChain/RAGを活用したLLMシステムの開発」と「Streamlitを活用したWebアプリの開発」という、LLMを搭載したWebアプリ（LLMアプリ）を開発するための土台となる知識・スキルは習得済みですので、多少調べながらスムーズにコード解読を進められるかと思います。

以下は、今回の題材のソースコードを通じて新たに学べることを「本格的なLLMアプリにおけるソースコードの全体像・書き方」と「具体的な技術要素」に分けて一覧化したものです。

いずれも重要なものですし、ソースコードの解読に取り組む上でも参考になる情報です。まずは今回の題材のソースコードを通じて新たに学べることの全体像を把握してもらえればと思います。

本格的なLLM開発における実践的なテクニック

[前章](#) [一覧に戻る](#) [次章](#)

項目	説明・補足
本格的なLLMアプリ開発における実践的なテクニック	モジュール・関数分割やコメント付け、Streamlitの実践的なテクニック、例外処理やアプリで共通利用する変数の管理办法など、実務でも通用する実践的な開発テクニック
本格的なチャットアプリのコードの書き方	種類によらず、チャットアプリのコードには基本的な型がある

具体的な技術要素

項目

説明・補足

YELL 学習支援システム Chapter2：サンプルアプリ①「社内情報特化型生成AI検索アプリ」



事前準備 ▾

ノード間の接続構造	ノード間の接続構造を定義するための設定と実行方法
例外発生時の対応	ログ出力と、ユーザーへのメッセージ表示
RAGで外部参照するデータソースの前処理	RAGの検索精度を上げるために、データソースに対して前処理（加工）を施す
スピナー表示	LLMから回答が返ってくるまで、クルクルと回るコードアニメーション（スピナー）を表示する
固定の文字列や数値など、各種データの管理方法	固定の文字列や数値などの各種データは、専用のファイルでまとめて変数定義して管理するのが一般的
Streamlitの様々な機能	<ul style="list-style-type: none"> ・ラジオボタン ・スピナー ・区切り線 ・メッセージ表示時の装飾 ・メッセージ表示時のアイコン設定 ・ブラウザタブの表示文言 ・サイドバー ・処理の中止 ・バグ解消のための空エリア ・画面再描画時に指定のコードが再実行されないようにするための制御

項目	説明・補足
APIからの返却データを、複雑な条件分岐の上で画面上に表示する方法	OpenAI APIからの返却データを複雑な条件分岐の上で画面上に表示できるようになれば、他のAPIからの返却データを画面上に表示する際にも応用が効く

フォルダ構成

ここでは、今回のソースコードにおけるフォルダ構成を説明します。各フォルダ・ファイルの役割を理解することで、コード解読をスムーズに行えるだけでなく、機能改修・追加を行う際に「どのファイルのコードを改善・追加すればいいか」を手軽に把握できます。

前提として、「Lesson5: Python基本文法」で説明済みですが、拡張子が「.py」のファイルを「モジュール」と呼びます。

一つのファイルに全てのコードを詰め込むとコードが非常に長くなり、解読が困難になるため、機能・役割ごとにモジュールを分割し、このモジュールを組み合わせることでシステム開発を行うのが一般的です。

以下は、ソースコードのトップのフォルダ内にあらかじめ格納されているフォルダ・ファイルの一覧です。特に重要なファイル名には先頭に「●」を付けています。

名前	役割
●main.py	アプリのメイン処理が記述されたファイル
●initialize.py	アプリ起動時にのみ実行される初期化処理が記述されたファイル
●components.py	画面表示に関する関数を定義するファイル
●utils.py	画面表示以外の様々な機能の関数を定義するファイル
constants.py	各ファイルから参照するために、固定の数値や文字列などのデータを変数化してまとめたファイル
.env	プログラム上のどこからでも呼び出せる変数（環境変数: OpenAI APIキーなど）の定義を記述するファイル
requirements.txt	アプリを動作させるのに使用するパッケージ情報が一覧化されたファイル
.gitignore	Gitのバージョン管理の対象外とするファイル・フォルダ名を記述するファイル
「data」 フォルダ	RAGの外部参照先のデータ置き場

名前	役割
「.streamlit」 フォルダ	Streamlitアプリのカスタマイズ・環境設定用のファイルを格納するフォルダ
「env」 フォルダ	Pythonの仮想環境フォルダ (名前は任意)

また以下は、プログラム実行時に作成されるフォルダの一覧です。

フォルダ名	役割
logs	このフォルダ内の「application.log」ファイルに、プログラムの実行ログが出力される。
__pycache__	Pythonプログラムを実行した際に自動作成されるキャッシュフォルダ (キャッシュを参照することで、プログラムの実行速度を上げられる)
.chroma	「chromadb」パッケージを使って作成されたベクターストアの実体

特に重要なのは、以下4つのファイルです。

(再掲)

名前	役割
●main.py	アプリのメイン処理が記述されたファイル
●initialize.py	アプリ起動時にのみ実行される初期化処理が記述されたファイル
●components.py	画面表示に関する関数を定義するファイル
●utils.py	画面表示以外の様々な機能の関数を定義するファイル

「streamlit run main.py」 のコマンドを実行して Streamlit アプリを起動すると、まず 「main.py」 のコードが実行されます。このファイルには、アプリのメイン処理が記述されています。

main.pyを実行すると、アプリ起動時に一度だけ 「initialize.py」 内のコードが実行されます。このファイルには初期化処理が記述されており、ログ設定や共通利用の変数設定、またRAGのChain作成などが行われます。

「components.py」 には、画面表示に特化した関数が定義されています。このファイル内の関数は全て、main.pyから呼び出されるものです。

「utils.py」には、画面表示以外の様々な機能の関数が定義されています。今回のアプリでは、このファイル内の関数も全てmain.pyから呼び出されます。

「initialize.py」「components.py」「utils.py」内で定義された関数は全て「main.py」から呼び出されることからも分かる通り、「main.py」のコードを見れば処理の全体像を把握できます。

補足として、今回のアプリではこの構成としていますが、この構成が正解というわけではありません。ファイル名やフォルダ名、構成には様々なパターンがありますので、あくまで一つの参考例として見てもらえばと思います。

コード解読のポイント

長くて複雑なコードを解読するにあたり、重要なポイントは以下の7点です。

1. どんなアプリ・システムかを理解する
2. コードの全体像を把握する
3. ブロックごとに分けて読む
4. 分からない部分は検索や生成AIツールを活用して調べる
5. printやログ出力を活用し、デバッグする
6. コードを書き換えると、処理や画面表示がどう変わるかを確認する
7. Google Colaboratory上で動作確認を行う
8. 根気強く読む

それについて、順に解説します。

1. どんなアプリ・システムかを理解する

まず大前提として、これから読むソースコードによって動作するアプリ・システムが、どんなものであるかをしっかり理解するべきです。様々な操作を試してみて、どんな機能があるかを一通り理解しましょう。

例えば今回のアプリにおいて、画面の観察、操作によって把握できる機能としては以下のようないことがあります。

- タイトル表示
- ブラウザタブの文言表示
- 利用目的を選択するラジオボタンの表示
- 初期画面におけるAIメッセージの表示
- 各利用目的の説明と入力例の表示
- チャット欄からメッセージを送信する機能
- チャット欄から入力したメッセージを表示する機能

- 入力に対してLLMからの回答を表示する機能
- LLMから回答が返ってくるまで、グルグル回るアニメーション（スピナー）を表示する機能
- 「社内文書検索」の回答モードにおいて、メインの文書ありかと、サブの文書ありかを表示する機能
- 「社内問い合わせ」の回答モードにおいて、参照した文書のありかを表示する機能

アプリ・システムについての理解を深めることで、「このコードはあの機能を実現するためのコードなのかな」といったように推測でき、コードが読みやすくなります。

2. コードの全体像を把握する

いきなり一つ一つのコードを調べながら細かく解読する前に、まずはコードの全体像を把握しましょう。これには、フォルダ構成・各ファイルの役割の理解も含まれます。

今回のアプリでは、メイン処理が「main.py」に記述されています。Pythonアプリでは、メイン処理を「main.py」や「app.py」といったファイルに記述することが多いです。

「main.py」のコードを上から順に「このブロックではこんな処理が行われているのかな」と推測しながらブロック単位で読むことで、コードの全体像を把握できます。

3. ブロックごとに分けて読む

コードを読む際は、全てのコードにいきなり細かく目を通す必要はありません。全体像の把握と重なっている部分もありますが、まずはブロックごとに「このブロックではだいたいこんな処理が行われているんだろう」といったように、ブロック単位で処理の大まかな概要を理解するようにしましょう。

機能改修・追加を行いたいと思った時に、該当箇所のコードを探して一つ一つコードを細かく読み、理解すれば問題ありません。

(ただし、今回のソースコードについては学習目的で用意しているものであるため、最終的には全てのコードの解読に取り組んでもらえればと思います。)

4. 分からない部分は検索や生成AIツールを活用して調べる

今回のソースコード内には、これまでのLessonで学習していない技術が数多く出てきます。後ほど解説しますが、細かい部分で分からないところがあったら、検索エンジンでの検索や、ChatGPTなどの生成AIツールを活用して調べましょう。

これまでのLessonでも、この「分からないことがあったら調べる」ということは何度も行ってきたかと思います。「学んだことがないからどうしても分からない」ではなく、根気強く調べるようにしましょう。

5. print関数やログ出力を活用し、変数の中身を確認する

「コード実行中に、この変数にはどんな値が入っているんだろう」と調べたいことは多々あるかと思います。そのような場合は、これまでにGoogle Colaboratory上でも行ってきたように、print関数を活用して中身を確認してみましょう。

print関数で出力を行うと、出力結果がターミナル上に表示されます。ターミナル上に出力されたものはGoogle Colaboratoryの出力結果と比べて見づらく感じるかもしれません。

そこで、今回新たに学ぶ「ファイルへのログ出力」を活用するのも一つの手です。ログ出力の専用ファイルに変数の中身を出力することで、より効率的にデバッグできるかもしれません。

6. コードを書き換えると、処理や画面表示がどう変わるかを確認する

「このコードで何が行われているか、どうしても分からない」という場面があるかもしれません。このような場面では、該当箇所のコードを書き換えて、処理や画面表示がどう変わるかを確認するのがおすすめです。

既存コードを積極的に書き換えてみて、Before-Afterを確認することは処理内容を理解するために効果的な手段の一つです。

7. Google Colaboratory上で動作確認を行う

VSCode上のデバッグに難しさを感じる場面があるかもしれません。そのような場合は、コードをGoogle Colaboratoryファイルのセルにコピーし、動作確認を行うのもおすすめです。

Google Colaboratoryのほうが対話形式でコード実行ができる分、変数の中身や処理の内容が把握しやすい場合もあります。

しかしStreamlitのコードはGoogle Colaboratory上では動作しないため、全てのコードをコピーするのではなく、Google Colaboratory上でも動作する細かいコード単位でコピーしましょう。

8. 根気強く読む

最後に、「根気強く読む」という点についてお伝えします。今回のコードは、本格的なLLMアプリとしては極めて簡単な部類です。次からの3つのChapterで扱う題材についても同じくです。

企業で新たに生成AIを活用したシステム開発を行う場合や、既存システムに生成AIの機能を実装する場合、また個人開発で多機能かつ複雑なLLMアプリを開発する場合、今回扱うコードとは比べ物にならないぐらいの量のコードになります。

長くて複雑なコードを読む際は、分からぬことだらけの場合が多いです。そのため、時間をかけてでも根気強く読むというのが結局のところ重要になります。

今回と、次のChapterからの3つのアプリにおいても、「根気強く読む必要がある」ということを念頭においてコード解説に取り組んでもらえればと思います。

処理の全体像

ここでは、処理の全体像について解説します。改めて、以下が特に重要なファイルです。

名前	役割
main.py	アプリのメイン処理が記述されたファイル
initialize.py	アプリ起動時にのみ実行される初期化処理が記述されたファイル
components.py	画面表示に関する関数を定義するファイル
utils.py	画面表示以外の様々な機能の関数を定義するファイル

メイン処理「main.py」のコードの流れを理解することで、処理の全体像を把握できます。

「components.py」と「utils.py」は、main.pyから呼び出される関数が定義されています。そのため、main.pyでコードの流れを理解しつつ、main.pyから呼び出される各関数の内容を理解することで、コードの全てを解説できます。

今回は初めて触れる本格的なLLMアプリということもあり、全てのコードにコメントを付けています。コード自体が教科書のようになっているため、コードだけでもある程度理解はできるかと思いますが、補足として解説にもお目通しいただければと思います。

メイン処理 (main.py)

まずはアプリのメイン処理が記述された「main.py」のコードについて解説します。

全体像

main.pyは、以下の「7つ」のブロックで構成されています。

- 1.ライブラリの読み込み
- 2.設定関連
- 3.初期化処理
- 4.初期表示
- 5.会話ログの表示
- 6.チャット入力の受け付け
- 7.チャット送信時の処理
 - 7-1.ユーザーメッセージの処理
 - 7-2.LLMからの回答取得
 - 7-3.LLMからの回答表示
 - 7-4.会話ログへの追加

上から順にコードを追っていくことで、LLMアプリの全体像を把握できます。

各ブロックの解説

各ブロックについて、順に解説します。

1. ライブラリの読み込み

```
# 「.env」ファイルから環境変数を読み込むためのモジュール
from dotenv import load_dotenv

# ログ出力を行うためのモジュール
import logging

# streamlitアプリの表示を担当するモジュール
import streamlit as st

# (自作) 画面表示以外の様々な関数が定義されているモジュール
import utils

# (自作) アプリ起動時に実行される初期化処理
from initialize import initialize

# (自作) 画面表示系の関数が定義されているモジュール
import components as cn
```

```
# (自作) 変数(定数)がまとめて定義・管理
import constants as ct
```

ここでは必要なライブラリ式をimportしています。各モジュールや関数については、コメントに記載の通りです。

2. 設定関連

```
# ブラウザタブの表示文言を設定
st.set_page_config(
    page_title=ct.APP_NAME
)

# ログ出力を行うためのロガーの設定
logger = logging.getLogger(ct.LOGGER_I
```

streamlitの「set_page_config」を使うことで、ブラウザタブの表示文言を設定できます。表示文言は、「constants.py」で定義された「APP_NAME」変数を参照しています。

またloggingモジュールの「getLogger」により、ログ出力を行うためのロガーの設定を行っています。ロガーとは、ファイルへのログ出力を担当するオブジェクトのことです。このコードを実行することにより、後続のコード実行中にエラーなどが発生した際、エラー内容が指定のログファイルに出力されるようになります。

このコードは、コード実行中にエラーなどが発生した際にログ出力が自動で行われるよう、各モジュール、また各関数内で呼び出す必要があります。

複数のモジュール・関数内で同じコードが実行されますが、引数で名前を設定することにより、再度同じコードを実行するとロガーが新規作成されるのではなく、過去に同じ名前で作成済みの場合は読み込みが行われるようになります。

3. 初期化処理

```
try:
    # 初期化処理（「initialize.py」の「initialize()」
except Exception as e:
    # エラーログの出力
    logger.error(f"{ct.INITIALIZE_ERR} {e}")
    # エラーメッセージの画面表示
```

```
st.error(utils.build_error_message())
```

```
# 後続の処理を中断  
st.stop()
```

```
# アプリ起動時のログファイルへの出力  
if not "initialized" in st.session_state:  
    st.session_state.initialized = True  
    logger.info(ct.APP_BOOT_MESSAGE)
```

「initialize.py」モジュールの「initialize」関数に、初期化処理のコードが記述されています。この初期化処理はアプリ起動時に一度だけ実行されます。

初期化処理として、大きく以下3つの処理が実行されます。

1. ログ出力の設定
2. 初期化データの用意
3. Retriever（ベクターストアからの検索を担当するオブジェクト）の作成

細かくコメントを付けていますため、各コードについて調べながら解説してもらえばと思います。

初期化処理の実行中に例外が発生すると、エラーログの出力とエラーメッセージの画面表示が行われます。

エラーログの出力には「logger.error()」を使います。引数には別ファイルで用意した変数を指定しています。

エラーメッセージの画面表示には「st.error()」を使います。引数には、様々な関数が定義された「utils.py」モジュールの「build_error_message()」関数の戻り値を渡しています。この関数では、エラーメッセージの整形が行われます。

またアプリ起動時に、アプリが起動されたことをログファイルに記録して残すための出を行っています。「st.session_state」内の辞書には「initialized」属性が定義されていないため、アプリ起動時に実行されます。if文の中で「initialized」属性の値にTrueを設定することで、再描画時にはこのif文の中身は実行されません。

エラー以外の、単純な情報の記録には「logger.info()」を使います。引数には、別ファイルで定義された変数を指定しています。

4. 初期表示

```
# タイトル表示  
cn.display_app_title()
```

```
# モード表示  
cn.display_select_mode()  
  
# AIメッセージの初期表示  
cn.display_initial_ai_message()
```

画面表示系の関数が定義された「components.py」を「cn」という名前で使えるようimport済みです。

このモジュールの「タイトル表示」「モード（利用目的）表示」「AIメッセージの初期表示」を行う関数をそれぞれ呼び出しています。

これにより、画面上部にタイトルが大きく表示され、またモード（利用目的）を選択するラジオボタン、AIメッセージの案内文が表示されます。

各関数の中身については、該当の関数のコードを読み込んでもらえばと思います。

5. 会話ログの表示

```
try:  
    # 会話ログの表示  
    cn.display_conversation_log()  
  
except Exception as e:  
    # エラーログの出力  
    logger.error(f"{ct.CONVERSATION_LI  
  
    # エラーメッセージの画面表示  
    st.error(utils.build_error_message(e))  
  
    # 後続の処理を中断  
    st.stop()
```

チャットでLLMとやり取りを行った後、再度チャット欄から送信するとコードが上から全て再実行され、過去のチャットが消えてしまいます。

そのため、LLMとのやり取りを行った後にユーザーメッセージとLLMレスポンスを会話ログ用のセッションに保存し、コードが再実行された際に会話ログの一覧が表示される処理を実装する必要があります。

components.pyモジュールの「display_conversation_log」が、会話ログの表示を担当している関数です。

例外処理が起きた際のコードは、先ほどの解説と同じものです。

6. チャット入力の受け付け

```
chat_message = st.chat_input(ct.CHAT_
```

チャットボットには、チャット入力欄の表示が必須です。「st.chat_input」を使うことで、チャット欄を表示できます。送信メッセージは変数「chat_message」に格納されますが、アプリ起動時はNoneという特殊な値（何も存在しないことを表す値）が格納されます。

7. チャット送信時の処理

```
if chat_message:  
    # 処理
```

チャット欄から送信を行うと「chat_message」に入力値が格納されるため、if文の条件式がTrueとなります。このif文の中では、チャット送信後の処理が記述されています。

7-1. ユーザーメッセージの処理

```
# ユーザーメッセージのログ出力  
logger.info({"message": chat_message,  
  
# ユーザーメッセージを表示  
with st.chat_message("user"):  
    st.markdown(chat_message)
```

LLMアプリでログとして記録を取りたい情報の一つが、LLMに渡されたユーザーメッセージです。チャットが送信された後、辞書型でユーザーメッセージと利用目的（「社内文書検索」or「社内問い合わせ」）の情報をログファイルに出力しています。

また画面にユーザーメッセージの表示を行っています。

7-2. LLMからの回答取得

```
# 「st.spinner」でグルグル回っている間、  
res_box = st.empty()  
  
# LLMによる回答生成（回答生成が完了するまで）  
with st.spinner(ct.SPINNER_TEXT):
```

```

try:

    # 画面読み込み時に作成したRetrie
    llm_response = utils.get_llm_

    except Exception as e:

        # エラーログの出力
        logger.error(f"{ct.GET_LLM_RE

        # エラーメッセージの画面表示
        st.error(utils.build_error_me

        # 後続の処理を中断
        st.stop()

```

utils.pyモジュールの「get_llm_response()」関数にユーザーメッセージを渡して実行することで、LLMからの回答が生成され、変数「llm_response」に格納されます。

LLMから回答が生成されるまではユーザーにとって待ち時間となります、画面に何も表示されずに数秒、場合によっては数十秒待たされると、ユーザーは「エラーなのか何のか」が分かりません。そのため、ユーザーの不安を解消するために、「スピナー」と呼ばれるグルグル回るアニメーションを表示するのが一般的です。

スピナーは、「with st.spinner()」で表示できます。このwith文の中のコード実行が完了するまで、スピナーが表示されます。引数として渡したテキストは、スピナーと一緒に表示されます。

またwith文の前に「st.empty()」というコードを実行していますが、これは Streamlitで空のエリアを表示するために用意しているものです。スピナーの with文の前に空のエリアを表示しておかないと、表示の不具合が発生することがあるため、スピナーのwith文と一緒に空のエリアを表示する必要があることを覚えておきましょう。

7-3. LLMからの回答表示

```

with st.chat_message("assistant"):

    try:

        # =====
        # モードが「社内文書検索」の場合

```

```

# =====
if st.session_state.mode == "c"

    # 入力内容と関連性が高い社内
    content = cn.display_search()

# =====
# モードが「社内問い合わせ」の場合
# =====
elif st.session_state.mode == "q"

    # 入力に対しての回答と、参照
    content = cn.display_content()

    # AIメッセージのログ出力
    logger.info({"message": content})

except Exception as e:

    # エラーログの出力
    logger.error(f"{ct.DISP_ANSEWR}: {e}")

    # エラーメッセージの画面表示
    st.error(utils.build_error_message(e))

    # 後続の処理を中断
    st.stop()

```

LLMからの回答が返ってきた後、AIメッセージとしてLLMからの回答を表示します。

表示に使う関数は、モード（利用目的）が「社内文書検索」か「社内問い合わせ」かによって分岐しています。モードによって表示形式が異なるためです。

7-4. 会話ログへの追加

```

# 表示用の会話ログにユーザーメッセージを追加
st.session_state.messages.append({"role": "user", "content": user_input})

# 表示用の会話ログにAIメッセージを追加
st.session_state.messages.append({"role": "assistant", "content": response})

```



最後に、「st.session_state」で、会話ログを追加するためのリスト「messages」にユーザーメッセージとAIメッセージをそれぞれ追加しています。

す。会話ログを表示する際にユーザーメッセージ用とAIメッセージ用で処理を分岐するため、「role」も一緒に渡しています。

ここまでが、メイン処理が記述された「main.py」のコードの全てです。メイン処理の流れを理解することで、LLMアプリの処理の流れの全体像を把握することができます。

提出課題

コードを解読した上で、「5つの必須課題」と「1つの任意（チャレンジ）課題」に取り組みましょう。

必須課題

レベル「易」の問題を2問、レベル「中」の問題を3問用意しています。それについて問題文を読み、機能改修・追加に取り組みましょう。

レベル「易」：2問

【問題1】

チャット送信後、ベクターストアから関連ドキュメントを取り出すRAGの処理が実装されていますが、現在、ベクターストアから取り出してプロンプトに埋め込む関連ドキュメントの数が「3」となっています。これを「5」に変更してください。

【問題2】

プログラミングにおいて、固定の数値を変数に格納せずそのまま使うことはおすすめできません。「その数値が何を表しているか」が分かりづらいため、固定の数値や文字列などは変数化するのが一般的です。

ちなみに、変数化されていない固定の数値を「マジックナンバー」と呼びます。マジックナンバーを使うことは、プログラミングにおいて非推奨とされています。

現在、「問題1」で変更した数値と、ドキュメントのチャンク分割を行う際に指定する「chunk_size」、また「chunk_overlap」の数値が、マジックナンバーとなっています。これらの数値を、適切なファイルで変数化して使うように修正してください。

レベル「中」：3問

【問題3】

現在の画面は、以下です。

社内情報特化型生成AI検索アプリ

社内文書検索
 社内問い合わせ

こんにちは。私は社内文書の情報をもとに回答する生成AIチャットボットです。上記で利用目的を選択し、画面下部のチャット欄からメッセージを送信してください。

【「社内文書検索」を選択した場合】

入力内容と関連性が高い社内文書のありかを検索できます。

【入力例】
社員の育成方針に関するMTGの議事録

【「社内問い合わせ」を選択した場合】

質問・要望に対して、社内文書の情報をもとに回答を得られます。

【入力例】
人事部に所属している従業員情報を一覧化して

こちらからメッセージを送信してください。 ➤

これを、以下の見た目になるようコードを修正してください。

社内情報特化型生成AI検索アプリ

社内文書検索
 社内問い合わせ

こんにちは。私は社内文書の情報をもとに回答する生成AIチャットボットです。サイドバーで利用目的を選択し、画面下部のチャット欄からメッセージを送信してください。

△ 具体的に入力したほうが期待通りの回答を得やすいです。

【「社内文書検索」を選択した場合】

入力内容と関連性が高い社内文書のありかを検索できます。

【入力例】
社員の育成方針に関するMTGの議事録

【「社内問い合わせ」を選択した場合】

質問・要望に対して、社内文書の情報をもとに回答を得られます。

【入力例】
人事部に所属している従業員情報を一覧化して

こちらからメッセージを送信してください。 ➤

これまでに学習していない技術を使いますので、Streamlitについて調べて実装する必要があります。

【問題4】

現在、「社内文書検索」と「社内問い合わせ」のいずれのモードにおいても、参照先のドキュメントのありかが表示されるようになっています。

ドキュメントのありかだけでなく、そのドキュメントの何ページ目を参照したかという情報がないと、例えばページ数が100ページ以上あるような場合に、どこを参照して回答したかの詳細は分かりません。

現状、PDFファイルのありかを表示する際、各モードにおいて以下のようにページ番号が表示されない状態となっています。

【「社内文書検索」モードでの回答】



株式会社EcoTeeの株主優待の内容を教えて



入力内容に関する情報は、以下のファイルに含まれている可能性があります。

./data/会社について/株主優待について.pdf

その他、ファイルありかの候補を提示します。

./data/会社について/会社概要.pdf

./data/サービスについて/デザインに関すること.pdf

【「社内問い合わせ」モードでの回答】

情報源

./data/会社について/会社概要.pdf

./data/サービスについて/デザインに関すること.pdf

./data/サービスについて/EcoTeeの代行出荷サービスについて.docx

各モードでPDFファイルのありかを表示する際、以下のようにページ番号も表示されるようにコードを修正してください。

【「社内文書検索」モードでの回答】



株式会社EcoTeeの株主優待の内容を教えて



入力内容に関する情報は、以下のファイルに含まれている可能性があります。

./data/会社について/株主優待について.pdf (ページNo.1)

その他、ファイルありかの候補を提示します。

./data/会社について/会社概要.pdf (ページNo.1)

./data/サービスについて/デザインに関すること.pdf (ページNo.3)

./data/顧客について/お客様情報.pdf (ページNo.4)

【「社内問い合わせ」モードでの回答】

情報源

./data/会社について/会社概要.pdf (ページNo.1)

./data/サービスについて/デザインに関する事.pdf (ページNo.3)

./data/サービスについて/EcoTeeの代行出荷サービスについて.docx

./data/サービスについて/Webサービス「EcoTee Creator」について.docx

./data/サービスについて/主要サービス・製品について.pdf (ページNo.2)

ページ番号を表示させるのはPDFファイルのみで問題ありません。

【問題5】

「data」フォルダ内の「MTG議事録」フォルダ内に、「議事録ルール.txt」というファイルが置かれています。

しかし現状、拡張子が「pdf」「docx」「csv」のファイルしかベクターストアに保存されない仕様となっています。

拡張子が「txt」のテキストファイルもベクターストアに保存されるよう、コードを修正してください。

任意（チャレンジ）課題

この課題への取り組みは任意です。提出の必要はありませんが、LLMの回答精度向上のための重要なスキル習得につながる課題内容ですので、余裕があればぜひ取り組んでみてください。

【問題6】

現在、「社内問い合わせ」モードにおいて「人事部に所属している従業員情報を一覧化して」という入力に対して、回答が以下のように返されます。

（※回答内容は人によって異なります。また「回答に必要な情報が見つかりませんでした。」と表示されることもありますので、毎回画面更新をした上で、何度

か試してみてください。)

④ 人事部に所属している従業員情報

社員ID	氏名 (フルネーム)	性別	生年 月日	年齢	メールアドレス	従業員区分	入社日	役職	スキルセット	保有資格	大学名	学部・学科	卒業年月日
EMP0014	佐藤 桃子	男性	1998-10-08	27	syoshida@example.com	契約社員	2018-12-08	主任	分析思考, 語学力, 人事管理, クラウドコンピューティング, UI/UXデザイン	基本情報技術者, 高度情報処理技術者	東北大	法学部	2021-05-03

現在、人事部に所属している従業員は佐藤 桃子さんのみです。

こちらからメッセージを送信してください。



上の例では人事部に所属している従業員が1人という回答が返ってきましたが、実際は9人います。つまり現状、「極めて回答精度が低い状態」です。

参照しているのは「data/社員について」フォルダ内の「社員名簿.csv」ですが、普通にCSVファイルを読み込むと各行ごとにドキュメントが分割されます。プロンプトに埋め込む関連ドキュメントの最大値は「5（問題1に取り組んだ後の数値）」であるため、最大でも5人までの情報しか一覧化されません。

このCSVファイルについて、各行に分割されたドキュメントを1つのドキュメントに統合し、検索精度が上がるようドキュメント内のテキストを調整した上で、「人事部に所属している従業員情報を一覧化して」という入力に対して4人以上の情報が一覧化されるようにコードを修正してください。

提出方法

1. ソースコード一式をGoogle ドライブにアップロードする

修正版の「company_inner_search_app」フォルダをGoogle ドライブの受講者用の以下フォルダ内にアップロードしてください。

【アップロード先のフォルダ】

「Lesson23: 【最終課題】各ユースケースごとのLLMアプリ開発」フォルダ

「Chapter2: サンプルアプリ①「社内情報特化型生成AI検索アプリ」」フォルダ

「提出用」フォルダ

【※アップロード時の注意事項】

修正版の「company_inner_search_app」フォルダの中には、アップロードする必要のないフォルダ・ファイルが含まれています。以下は、アップロード不要なフォルダの一覧です。

フォルダ名	役割
__pycache__	Pythonコードの実行を高速化するためのキャッシュフォルダ
.chroma	Chromaデータベースのフォルダ
env	Python仮想環境のフォルダ (人によっては別名の可能性あり)
.git	Gitによるファイル等の変更履歴が記録されているフォルダ

これらのフォルダが含まれていると、修正版の「company_inner_search_app」フォルダは容量が膨大になります。Googleドライブのアップロードに膨大な時間を要するため、削除してからアップロードしましょう。

上記のフォルダの中で、存在しないものがあるかもしれません。存在しないフォルダについての削除対応は不要です。

「.git」フォルダは、VSCodeのサイドバーから削除できない可能性があります。そのため、Windowsの場合はエクスプローラーから、Macの場合はFinderから直接削除してください。

また、以下のファイルも削除しましょう。

ファイル名	役割
.env	APIキーなどを環境変数として記述するファイル

「.env」ファイルにはAPIキーが記述されていますが、APIキーは他者に共有してはいけません。そのため、Googleドライブにフォルダをアップロードする前に必ず削除してください。

これらのフォルダ・ファイルを作業用の「company_inner_search_app」フォルダ内で削除すると、作成したアプリが動作しなくなります。

そのため、アップロード用に「company_inner_search_app」フォルダを複製し、複製したフォルダ内でアップロード不要なフォルダ・ファイルを削除しましょう。このアップロード用の「company_inner_search_app」フォルダをGoogleドライブにアップロードしてください。

【アップロード後のGoogle ドライブ上のフォルダについて】

MacのPCをお使いの場合、Google ドライブにフォルダをアップロードすると、内部の各フォルダ内に「.DS_Store」という名前のファイルが自動作成されます。このファイルには、フォルダ・ファイルに関する各種情報が記載されています。

「.DS_Store」ファイルが存在していてもアプリの動作に影響はないため、削除対応は不要です。

2. Slackから課題を提出する

アップロードした「company_inner_search_app」フォルダを右クリックして「共有」→「リンクをコピー」でフォルダのリンクをコピーし、Slackにて、課題提出フォームを用いて提出してください。

チャプターの学習を完了する

© 2014 - 2025 Infratop Inc.