日日是Oracle APEX

Oracle APEXを使った作業をしていて、気の付いたところを忘れないようにメモをとります。

2024年2月14日水曜日

OllamaでCode Llamaを実行しAPEXアプリケーションからコードを生成させる

Oracle APEX界隈で著名なPlamen Muskovさんが、X（旧Twitter）にてOllamaを使ってLLaVAを呼び出すAPEXアプリケーションを作ったと投稿していました。

Plamen Mushkov · 2月13日
@plamen_9 · フォローする
@ollama is now compatible with OpenAI Chat Completions API. Why is this excellent news?

Because you can run any kind of Open source LLMs on your local machine (and even your phone). And not only that - you can run your existing code, that is written using OpenAI's APIs, because...

> **ollama** @ollama
> Ollama OpenAI compatibility is here!
>
> ollama.ai/blog/openai-co...
>
> Time to update to 0.1.24! Let's go!

Plamen Mushkov
@plamen_9 · フォローする

Not convinced yet?

Give my little POC a try - built in a single day, using the best Low-code tool - Oracle APEX (#orclAPEX) with some REST API calls to my local machine.

Now, explaining memes to you, using the latest version of LLaVA multi-modal LLM, run locally using Ollama ... さらに表示

午前5:46 · 2024年2月13日

5 返信 リンクをコピー

Xでもっと読む

Ollamaであれば簡単にCode Llamaを動かすことができそうなので、以前に作成した環境で試してみました。**Ollama**は**llama_cpp.server**と同様に、APIサーバーとして動作します。

llama_cpp.serverをAmpere A1のインスタンス上で動かしてみる
https://apexugj.blogspot.com/2023/07/llamacppserver-on-ap.html

以下はその作業の記録です。

OllamaのGitHubのページに、Ollamaのインストール手順が記載されています。

Always FreeのAmpere A1のコンピュート・インスタンスが作成済みとします。OSはUbuntuです。接続ユーザーは**ubuntu**（Oracle Linuxのインスタンスでのopcに当たります）になります。

ユーザーubuntuで接続し、**Linux & WSL2**のインストール手順として記載されている以下のコマンドを実行します。

```
ubuntu@mywhisper2:~$ curl -fsSL https://ollama.com/install.sh | sh
>>> Downloading ollama...
############################################################################
100.0%##############################################################################
100.0%
>>> Installing ollama to /usr/local/bin...
>>> Creating ollama user...
>>> Adding ollama user to render group...
>>> Adding current user to ollama group...
>>> Creating ollama systemd service...
>>> Enabling and starting ollama service...
Created symlink /etc/systemd/system/default.target.wants/ollama.service →
/etc/systemd/system/ollama.service.
```

```
>>> The Ollama API is now available at 0.0.0.0:11434.
>>> Install complete. Run "ollama" from the command line.
WARNING: No NVIDIA GPU detected. Ollama will run in CPU-only mode.
ubuntu@mywhisper2:~$
```

インストールを実行しているユーザーがroot権限を持っているか、sudoでroot権限を取れることが前提のようです。スクリプトの実行が完了すると、OSのサービスとして**ollama**がインストールされます。また**/usr/local/bin**以下に**ollama**が作成されます。

インストール直後に**ollama**がサービスとして起動します。サービスの起動と終了には**systemctl**コマンドを使用します。

モデル**codellama**を呼び出せるように、ダウンロードします。

```
ubuntu@mywhisper2:~$ ollama run codellama
pulling manifest
pulling 3a43f93b78ec... 100%  ▕▕▕▕▕▕▕▕▕▕▕▕▕▕▕▕  3.8 GB
pulling 8c17c2ebb0ea... 100%  ▕▕▕▕▕▕▕▕▕▕▕▕▕▕▕▕  7.0 KB
pulling 590d74a5569b... 100%  ▕▕▕▕▕▕▕▕▕▕▕▕▕▕▕▕  4.8 KB
pulling 2e0493f67d0c... 100%  ▕▕▕▕▕▕▕▕▕▕▕▕▕▕▕▕    59 B
pulling 7f6a57943a88... 100%  ▕▕▕▕▕▕▕▕▕▕▕▕▕▕▕▕   120 B
pulling 316526ac7323... 100%  ▕▕▕▕▕▕▕▕▕▕▕▕▕▕▕▕   529 B
verifying sha256 digest
writing manifest
removing any unused layers
success
>>> /?
Available Commands:
  /set            Set session variables
  /show           Show model information
  /load <model>   Load a session or model
  /save <model>   Save your current session
  /bye            Exit
  /?, /help       Help for a command
  /? shortcuts    Help for keyboard shortcuts

Use """ to begin a multi-line message.

>>> /bye
ubuntu@mywhisper2:~$
```

以上でCode Llamaを呼び出す準備ができました。

Ollamaはデフォルトで**TCP**ポート**11434**を開けて、REST APIの要求を待ち受けます。

firewalldに対して、TCPポート11434への接続を許可します。

```
ubuntu@mywhisper2:~$ sudo firewall-cmd --add-port=11434/tcp
You're performing an operation over default zone ('public'),
but your connections/interfaces are in zone 'docker' (see --get-active-zones)
You most likely need to use --zone=docker option.

success
ubuntu@mywhisper2:~$
```

変更を永続化します。

```
ubuntu@mywhisper2:~$ sudo firewall-cmd --runtime-to-permanent
success
ubuntu@mywhisper2:~$
```

リクエストをHTTPSで受け付けてHTTPでllama_cpp.serverを呼び出していたNginxの設定を変更します。**/etc/nginx/conf.d**以下の**server.conf**に含まれる**proxy_pass**の行のポート番号を8000から**11434**に変更します。

proxy_pass http://localhost:11434/;

以上の変更を行なった後にNginxを再起動すると、Ollamaを呼び出せる状態になります。

OllamaとCode Llamaを呼び出すAPEXアプリケーションについて、簡単に紹介します。

OllamaはOpenAIのchat completions互換APIをサポートしているとのことですが、今回はCode Llamaを呼び出すので、単純にGenerate a completionを呼び出すことにしました。

ページ・アイテム**P1_MODEL**に**モデル名**を設定します。今回は**codellama**の指定だけを想定していますが、一応変更できます。**P1_MESSAGE**に**プロンプト**となる文章を入力します。Generation APIの呼び出しに与える値は以上です。

ページ・アイテム**P1_RESPONSE**にCode Llamaの出力を表示します。**マークダウン**が返されるため、**アイテム・タイプ**に**Markdownエディタ**を選択しています。ページ・アイテム**P1_STATS**にAPI呼び出しに関する統計情報を表示します。

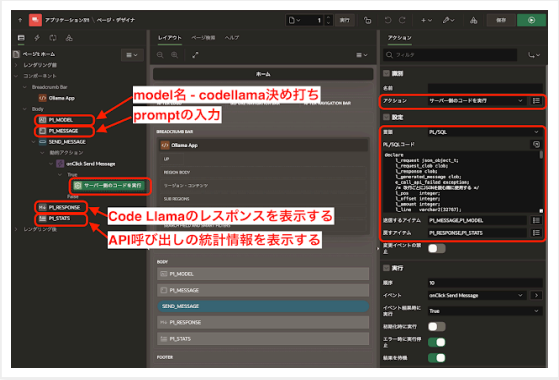
Ollama/Code Llamaの呼び出しは、ボタン**SEND_MESSAGE**に実装した**動的アクション**により実行します。**TRUEアクション**の**サーバー側のコードを実行**の**PL/SQLコード**として以下を記述します。Ollamaのレスポンスはデフォルトでストリーミングで、ストリーミング出力にしないとNginxがタイムアウトするため、APEX側でCode Llamaのレスポンスをつなげ合わせています。

```
declare
    l_request json_object_t;
    l_request_clob clob;
    l_response clob;
    l_generated_message clob;
    e_call_api_failed exception;
    /* 改行ごとにJSONを読む際に使用する */
    l_pos    integer;
    l_offset integer;
    l_amount integer;
    l_line   varchar2(32767);
    l_json   json_object_t;
    l_is_done boolean;
begin
    /*
     * OllamaのGenerate a completionを呼び出す。
     * https://github.com/ollama/ollama/blob/main/docs/api.md
     */
    l_request := json_object_t();
    l_request.put('model', :P1_MODEL);
```

```
l_request.put('prompt', :P1_MESSAGE);
l_request_clob := l_request.to_clob();
/*
 * REST APIの呼び出し。Always FreeのAmpere A1インスタンスは遅いので
 * p_transfer_timeoutの設定は必須。
 */
apex_web_service.clear_request_headers();
apex_web_service.set_request_headers('Content-Type', 'application/json', p_reset => false);
l_response := apex_web_service.make_rest_request(
    p_url => :G_ENDPOINT
    ,p_http_method => 'POST'
    ,p_body => l_request_clob
    ,p_transfer_timeout => :G_TRANSFER_TIMEOUT
);
if apex_web_service.g_status_code <> 200 then
    raise e_call_api_failed;
end if;
/*
 * 出力はデフォルトでストリーミングで、APEXの場合、Ollamaが分割で返しているレスポンスが
 * ひとつのレスポンスとして返される。そのため、レスポンスを改行ごとに分割して、それぞれを
 * JSONとして解釈する必要がある。
 */
l_generated_message := '';
l_offset := 1;
while true
loop
    /*
     * LFの位置を探す。
     * 返される位置はoffsetからの相対位置ではなく、CLOBの中での絶対位置が返される。
    */
    l_pos := dbms_lob.instr(
        lob_loc  => l_response
        ,pattern => CHR(10)
        ,offset  => l_offset
    );
    /* LFがなければ終了 */
    exit when ( l_pos = 0 );
    l_amount := l_pos - l_offset;
    /* 1行取り出す */
    l_line := dbms_lob.substr(
        lob_loc => l_response
        ,amount => l_amount
        ,offset => l_offset
    );
    l_json := json_object_t(l_line);
    l_generated_message := l_generated_message || l_json.get_string('response');
    l_is_done := l_json.get_boolean('done');
```

```
        if l_is_done then
            /* context以外をSTATSとして印刷 */
            l_json.remove('context');
            :P1_STATS := l_json.to_string();
        end if;
        /* 次の行を取り出す */
        l_offset := l_pos + 1;
    end loop;
    :P1_RESPONSE := l_generated_message;
end;
```

call-ollama-codellama-generate.sql hosted with ❤️ by GitHub view raw

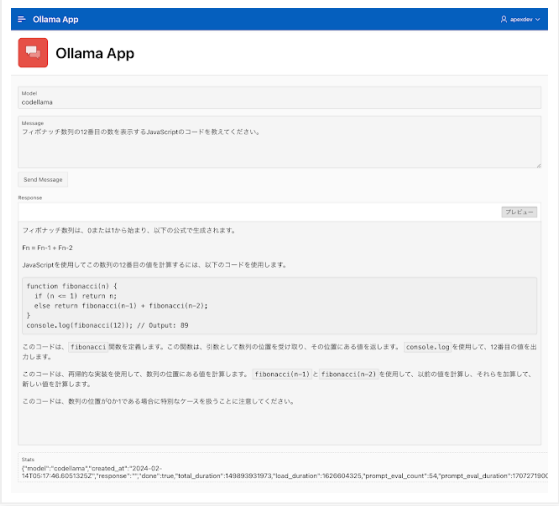


アプリケーション定義の**置換**に置換文字列**G_ENDPOINT**として、OllamaのAPI呼び出しのエンドポイント（**https://ホスト名/api/generate**）、**G_TRANSFER_TIMEOUT**にAPEX側のAPEX_WEB_SERVICE.MAKE_REST_REQUESTの**p_transfer_timeoutに与える秒数**を指定します。

以下、プロンプトとして以下を与えた結果です。

「フィボナッチ数列の12番目の数を表示するJavaScriptのコードを教えてください。」

レスポンスが返されるまで２分はかかるので実用性はありませんが、Code Llamaが日本語を受け付けてレスポンスも日本語で返してきたのには、少し驚きました。



今回作成したAPEXアプリケーションのエクスポートを以下に置きました。
https://github.com/ujnak/apexapps/blob/master/exports/ollama-codellama-app.zip

Oracle APEXのアプリケーション作成の参考になれば幸いです。

完

Yuji N. 時刻: 18:29

共有

ウェブ バージョンを表示

自己紹介

Yuji N.

日本オラクル株式会社に勤務していて、Oracle APEXのGroundbreaker Advocateを拝命しました。
こちらの記事につきましては、免責事項の参照をお願いいたします。

詳細プロフィールを表示