

いろいろなツールを
使いこなそう!

連載

清水 美樹 ▶ 初心者向けのプログラミング解説を専門とする、Macが好きなフリーのテクニカルライター。専門書の翻訳も手掛ける。工学博士(東北大学)。

Macで楽しい プログラミング

第3回 Appleシリコンを生かしてオープンソースLLMを動かす

最近、AIと言えば「生成AI」ですね。中でも、自然な文章を生成できる「LLM」(Large Language Model、大規模言語モデル)と、文章から画像を生成できる「画像生成AI」が大きな注目を集めています。

Appleシリコン搭載Macは生成AIに向く

生成AIを動かすシステムの世界では、米NVIDIA製のGPUが目下大活躍していますが、実は我らがMacも比較的、生成AIのシステムに向くコンピュータと考えられています。本家本元の米Appleも、MacやiPhoneのローカル環境で動く生成AIの「Apple Intelligence」を発表し、2024年内に一部の機種から提供を開始する予定です。

Macが生成AI向きなのは、近年のMacが搭載する、GPU内蔵の高性能SoC(System on a Chip)である「Appleシリコン」(M1、M2、M3等)と、GPUのビデオメモリー(VRAM)としても使える「ユニファイドメモリー」の組み合わせが生成AIの実行に適しているからです。Appleシリコン搭載のMacなら、8GBのユニファイドメモリーしか内蔵していない下位モデルでも、生成AIをそこそこ快適に動かせます。

Macユーザーなら、ぜひ生成AIにチャレンジしてみましようということで、今回はMacのローカル環境でオープンソースのLLMを動かす方法を紹介します。

オープンソースLLMとOllama

今、「ローカル環境でオープンソースのLLMを動かす」と書きましたが、この意味をもう少し細かく説明しましょう。

まず、LLMは、膨大な文章を学習したディープニューラルネットワークの一種です。そして、学習済みのディープニューラルネットワークの状態を表すパラメータをファイルに保存することで、そのLLMを保存し、配布することもできます。

ChatGPTなどのLLMでは、LLMのパラメータのファイルが公開されていません。一方で、オープンソースの形で公開されているLLMのパラメータのファイルはたくさんあります。代表例は、米Metaが公開している「Llama 3」や米Googleが公開している「Gemma 2」で、これらのLLMのパラメータのファイルは、容易にダウンロードできます。本稿では、後ほどLlama 3のファイルを入手して利用してみます。

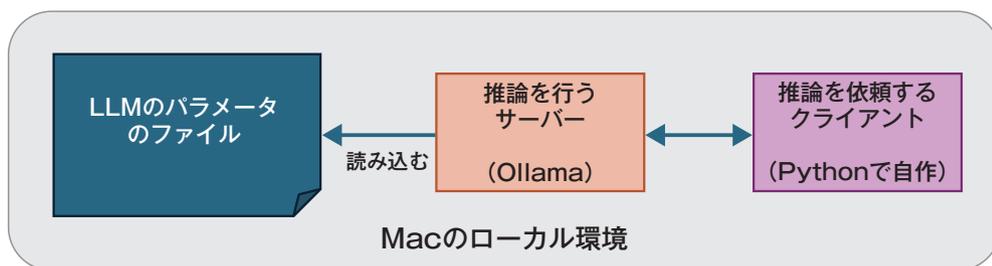
LLMのパラメータのファイルは、データベースのファイルと同じで、それだけでは動きません。ファイルを読み込んで、チャットなどのために“推論”を行うプログラムが必要です。

そのプログラムは、単独で動く形のもの、クライアント/サーバーの形で動くものに分けられます。今回は後者の、クライアント/サーバーの形で動くプログラムを作成することにします。

つまり、オープンソースLLMのパラメータのファイルと、そのファイルを読み込んで推論を行うプログラムをすべてローカル環境で動かすことが、「ローカル環境でオープンソースのLLMを動かす」の意味になります。

さて、今回はクライアント/サーバーの形で動くプログラムを作成すると書きましたが、サーバーは、オープンソースで提供されている「Ollama」というソフトウェアを使うことにします。ですから、実際に作成するの

図1 ● LLMのファイル、Ollama(推論を行うサーバー)、推論を依頼するクライアントのすべてがMacのローカル環境にインストールされている。クライアントのPythonプログラムがOllamaのサーバーに推論を依頼すると、LLMのファイルを読み込んだOllamaのサーバーが推論を行い、返答をクライアントに返す

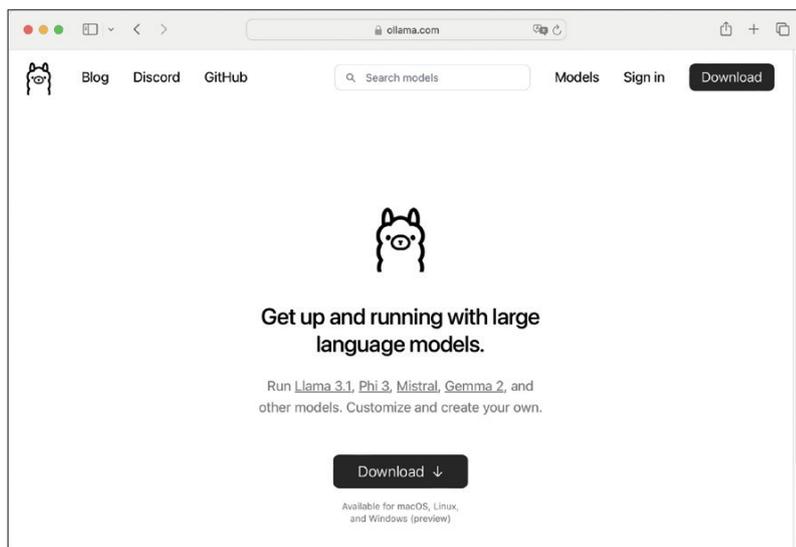


スします。

<https://ollama.com/>

表示されている「Download ↓」(図2) → 「Download for macOS」をクリックすると、「Ollama」(Ollama.app)とい

図2 ● Ollamaの公式サイトからOllamaをダウンロードする



うアプリケーションのファイルをダウンロードできます(図3)。これを「アプリケーション」のフォルダにドラッグ&ドロップしましょう。

次に、Ollamaを起動します。初回の起動時には図4の画面が表示されるので、画面に従って、Ollamaのコマンドである「ollama」をインストールします。このollamaは「/usr/local/bin/ollama」にインストールされます。これでollamaコマンドをターミナルで使えるようになります。

Llama 3を使ってみる

は、Ollamaに接続するクライアントのプログラムだけです(図1)。クライアントのプログラムはPythonで作ります。

図4の最後の画面に、「好きなターミナルで『ollama run llama3』のコマンドを実行してください」と書いてありますから、ターミナルを起動して、このコマンドを実行してみます。すると、Llama 3のファイルのダウン

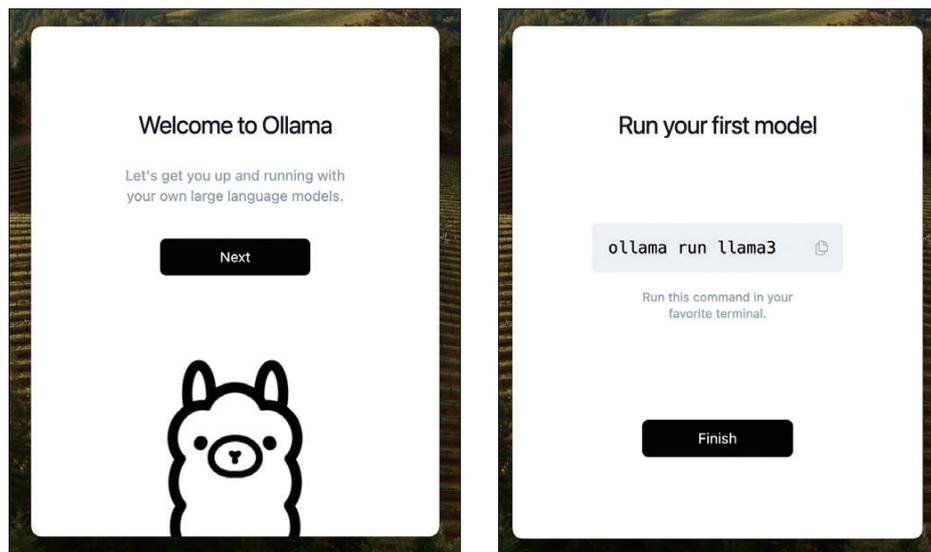
Ollamaの導入

では、サーバーとなるOllamaの導入から始めましょう。Ollamaの公式サイトにアクセス

図3 ● ダウンロードしたOllamaのアプリケーション



図4 ● 初回起動時に、画面に従って、Ollamaのコマンドである「ollama」をインストールする



ロードが始まります。約4.7GBのファイルをダウンロードするので完了までに少し時間がかかりますが、ダウンロードが完了すると、自動的にLlama 3のLLMとチャットできるようになります。「>>>」というプロンプトの後に、英語や日本語の文章を入力して何か質問すると、すぐに返答が表示されます。

図5は、「Please show me three interesting points of programming.」(プログラミングの面白い点を3つ示してください)と質問した例です。「停止性問題」「コードの臭い」の概念「抽象化の力」の3つを示してくれました。

図6は、日本語で「最も人気のあるプログラミング言語を教えてください」と尋ねた結果で、JavaScriptをその理由とともに挙げてくれました。LLMの返答が常に正しいとは限りませんが、図5と図6の返答は、いずれも適切な内容だと思われます。なお、Macに付属する「アクティビティモニタ」で推論中(返答文の生成中)の様子を観察すると、Ollamaのサーバーのプロセスである「ollama_llama_server」のGPU使用率が非常に高くなるのがわかります(図7)。ちゃんとAppleシリコンのGPUがLLMの処理を担っていること

図5●Llama 3に「Please show me three interesting points of programming.」(プログラミングの面白い点を3つ示してください)と質問した例

```

ken -- ollama run llama3 -- 80x40
Last login: Sat Jul 27 07:54:09 on ttys000
ken@Ken ~ % ollama run llama3
pulling manifest
pulling 6a0746a1ec1a... 100% ██████████ 4.7 GB
pulling 4fa551d4f938... 100% ██████████ 12 KB
pulling 8ab4849b038c... 100% ██████████ 254 B
pulling 577073ffcc6c... 100% ██████████ 110 B
pulling 3f8eb4da87fa... 100% ██████████ 485 B
verifying sha256 digest
writing manifest
removing any unused layers
success
>>> Please show me three interesting points of programming.
Here are three interesting points about programming:

1. The halting problem: In 1936, mathematician Alan Turing showed that there cannot exist an algorithm that can determine, given an arbitrary program and input, whether the program will run forever or eventually stop. This means that it's impossible to write a program that can predict with certainty whether another program will terminate or not. This has important implications for programming languages and their design.

2. The concept of "code smells": Code smells are patterns in code that indicate poor design, unnecessary complexity, or potential bugs. For example, if a function is doing too many things at once, it might be considered a "long method smell." Identifying and refactoring code smells can improve the maintainability, readability, and performance of software.

3. The power of abstractions: Programming involves creating abstract representations of real-world problems or systems, using concepts like variables, loops, functions, and data structures. Abstraction allows programmers to model complex phenomena in a simplified way, making it easier to reason about and manipulate the code. This ability to abstract away from details is what enables us to write software that can solve real-world problems in a scalable and maintainable manner.

I hope you find these points interesting!
  
```

図6●Llama 3に「最も人気のあるプログラミング言語を教えてください」と質問した例

```

>>> 最も人気のあるプログラミング言語を教えてください。
😊

最も人気のあるプログラミング言語は、JavaScriptです。理由として以下のような点が挙げられます。

1. Web開発: JavaScriptは、ウェブページに組み込むことができ、インタラクティブな体験を提供するために使用されます。
2. モバイルアプリケーション: JavaScriptは、React NativeやIonicなどのフレームワークでスマートフォンアプリケーションの開発に用いられます。
3. サーバーサイドプログラミング: Node.jsを使用したJavaScriptは、サーバーサイドプログラミングにも使われます。
4. Machine Learning: TensorFlowやBrain.jsなどのフレームワークでJavaScriptは機械学習の開発にも用いられます。

このため、JavaScriptは、最も人気のあるプログラミング言語の1つとなっています。
  
```

を確認できます。

利用しているLlama 3の情報を表示してみましょう。「>>>」のプロンプトの後に「/show info」と入力します。すると、図8の情報が表示されます。ここで注目したいのは、「parameters」と「quantization」の項

図7 ●ollama_llama_serverのCPUの使用率(% CPU)は低いが、GPUの使用率(% GPU)は非常に高くなっている

プロセス名	% CPU	CPU時間	スレッド	アイドル状態解除	種類	% GPU	GPU時間	PID	ユーザ
ollama_llama_server	2.4	1.02	13	0	Apple	98.2	4.85	2400	ken
WindowServer	10.4	6:47.85	15	167	Apple	0.6	50.74	362	_windowserver
ターミナル	6.0	2:58.39	10	25	Apple	0.2	7.67	1340	ken
kernel_task	9.3	12:57.09	477	1659	Apple	0.0	0.00	0	root

目です。

parametersは、LLMのディープニューラルネットワークの状態を表すパラメータ数で、この数が大きいほど、高性能なLLMだと言えます。ただし、パラメータ数が大きくなるほど、そのパラメータを記録するファイルのサイズも大きくなり、大容量のユニファイドメモリを搭載するMacでしか動かなくなります。図8では、parametersが「8.0B」(Bはbillionの意味)になっているので、このLlama 3は80億パラメータ版であることがわかります。実は、Llama 3には、8B版(80億パラメータ版)と、より高性能な70B版(700億パラメータ版)があります。

quantizationは、「量子化」のビット数を示します。パラメータのファイルは巨大です。そこで、量子化という処理を施して、ファイルのサイズを圧縮することがしばしば行われます。図8では、quantizationが「Q4_0」になっています。これは、4ビットで量子化されたLlama 3のファイルであることを意味しています。量子化のビット数が小さいほどファイルのサイズは小さくなりますが、性能は劣化します。

チャットを終了してターミナルのシェルに戻るには、「/

bye」と入力します。ただし、チャットを終了しても、Ollamaのサーバーは常駐して動いています。Ollamaのサーバーも終了したい場合は、メニューバーに表示されているOllamaのアイコンから「Quit Ollama」を選びましょう。なお、Ollamaをインストールすると、ログイン時にOllamaのサーバーが起動するようになります。このログイン時の起動を止めたいのであれば、Macの「システム設定」→「一般」→「ログイン項目」から「Ollama」を削除しましょう。

最新のLlama 3.1 を使ってみる

実は、Llama 3は、2024年7月23日に公開された「Llama 3.1」が、原稿執筆時点(2024年7月末)では最新版です。OllamaではLlama 3.1を簡単に導入できるので、やってみましょう。先ほどと同じような次のollamaコマンドをターミナルで実行するだけです。

```
ollama run llama3.1
```

約4.7GBのファイルのダウンロードが完了したら、チャットができます。試しに、「誕生日をお祝いする文章を示してください」という質問文を入力したら、「今日はあなたの誕生日です。心からおめでとう!」という返答文が生成されました(図9)。

「/show info」と入力してダウンロードしたLlama 3.1の情報を表示すると、parameters(パラメータ数)は8.0B(80億)で、quantization(量子化)はQ4_0(4ビット)だとわかります。ちなみに、Llama 3.1には、8B版と70B版、405B版(4050億パラメータ版)があります。このうち、最大の405B版は現時点で最高性能を持つLLMの一つと考えられています。

ところで、Ollamaでダウンロードした

図8 ●Llama 3の情報を表示

```
>>> /show info ← 入力
Model
  arch          llama
  parameters    8.0B
  quantization  Q4_0
  context length 8192
  embedding length 4096

Parameters
  num_keep      24
  stop          "<|start_header_id|>"
  stop          "<|end_header_id|>"
  stop          "<|eot_id|>"

License
  META LLAMA 3 COMMUNITY LICENSE AGREEMENT
  Meta Llama 3 Version Release Date: April 18, 2024
```

LLMのファイルは、ホームフォルダの「.ollama/models/blobs」フォルダにハッシュ値のファイル名で格納されています。ですから、どれがどのLLMのファイルなのかは簡単にはわかりません。

ダウンロードしたLLMの削除には、ollamaコマンドを使います。Llama 3.1をダウンロードしたので、Llama 3は削除しておきます。次のコマンドを入力します。

```
ollama rm llama3
```

実行すると、「deleted llama3」と表示されます。

クライアントのプログラムを作る

最後に、OllamaのサーバーとLlama 3.1に質問文を送信できるクライアントのプログラムをPythonで作成してみます。まず、Pythonの「ollama」ライブラリを、次のコマンドでインストールします。

```
pip3 install ollama
```

次に、Ollamaが起動していることを確認したら、クライアントのプログラムであるリスト1の「birthday.py」を作成します。このプログラムでは、ollamaライブラリを利用して、「誕生日をお祝いする文章を示してください」という質問文をOllamaのサーバーへ送ります。

birthday.pyは次のコマンドで実行します。

```
python3 birthday.py
```

図9 ● Llama 3.1をダウンロードして、チャットを試す

```
ken@ken ~ % ollama run llama3.1
pulling manifest
pulling 87048bcd5521... 100% ██████████ 4.7 GB
pulling 8cf247399e57... 100% ██████████ 1.7 KB
pulling f1cd752815fc... 100% ██████████ 12 KB
pulling 56bb8bd477a5... 100% ██████████ 96 B
pulling e711233e7343... 100% ██████████ 485 B
verifying sha256 digest
writing manifest
removing any unused layers
success
>>> 誕生日をお祝いする文章を示してください
今日はあなたの誕生日です。心からおめでとう！

>>> end a message (? for help)
```

リスト1 ● 「birthday.py」。Ollamaのサーバーに推論(質問文への返答の生成)を依頼するクライアントのPythonプログラム

```
import ollama

response = ollama.chat(model='llama3.1', messages=[
    {
        'role': 'user',
        'content': '誕生日をお祝いする文章を示してください'
    }
])
print(response['message']['content'])
```

図10 ● リスト1の実行例

```
ken@ken ~ % python3 birthday.py
例文：
「今日はお誕生日です！この日のために、プレゼントを手作りしてきたのですが、喜んで頂けると嬉しいです。家族と一緒に過ごす時間がもてますよう楽しみにしています。」

また、「お誕生日おめでとうございます！今年もあなたにとって素敵な年になりますように願っています。たくさんの喜びを送ります。」などと書くことも可能です。
```

しばらく待つと、図10の文章(質問文に対する返答)が表示されます。

リスト1では、「ollama.chat関数」を実行し、その戻り値を表示しています。ollama.chat関数の引数modelには、利用するLLMの名前を文字列で指定します。引数messagesには、LLMへ送信する質問文を、Pythonのリストと辞書を組み合わせたデータ構造内の'content'キーの値として記述します。

ollama.chat関数を実行すると、Ollamaのサーバーへ質問文が送られ、しばらくすると、LLMが生成した返答の文章が戻り値のresponseに格納されます。リスト1では最後に、「print(response['message']['content'])」で、その文章を表示しています。🔗