

タブ 1

GPU無しでOllamaの動作確認

目標は、デイサービスで働くロボット。おもしろ人形を納品し、おもしろパソコンを回収してきた。回収したおもしろパソコンのAIをggmlからOllamaに変更する。まずは、動作確認を行う。

構成

・W8yellow\$	Client PC CF-W8	Debian12
・Ryzen9\$	Server PC Minis Forum HX90	Debian12

目次

[概要](#)

[Ollamaで生成AIの実行](#)

[Ollamaのインストール](#)

[docker run](#)

[モデルのダウンロード](#)

[Ollamaのcuiでの実行方法](#)

[Ryzen9に接続](#)

[Ollamaが起動しているか確認](#)

[Ollamaの実行](#)

[モデルの確認](#)

[質問: ros noeticでollama APIの利用方法](#)

[回答: Gemma3n:e4b](#)

[結論](#)

[今後の予定](#)

[参考](#)

概要

やはり、過去の記事を読んでも、しっくりこない。

ほぼ、忘れている。数ヶ月前の話であるが。。とにかく、Ollamaを起動して、gemma3:12bを動かしてみる。重ければ、下げていく。27bについては、GPU無しなので、テスト対象外とする。いや、最後にやるだけやってみるか？

Ollamaで生成AIの実行

Ollamaのインストール

```
yo@Ryzen9:~/command/13docker_ollama$ docker pull  
ollama/ollama
```

```
docker run
```

```
yo@Ryzen9:~/command/13docker_ollama$ cat  
./docker_run_ollama.sh  
#!/bin/bash  
docker run -d \  
-v ollama:/root/.ollama \  
-p 11434:11434 \  
--name ollama \  
ollama/ollama
```

```
yo@Ryzen9:~/command/13docker_ollama$ ./docker_run_ollama.sh
```

モデルのダウンロード

```
yo@Ryzen9:~/command/13docker_ollama$ docker exec -it ollama  
ollama pull gemma3:12b
```

Ollamaのcuiでの実行方法

Ryzen9に接続

```
yo@W8yellow:~$ ssh -X yo@192.168.11.8
```

Ollamaが起動しているか確認

```
yo@Ryzen9:~$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED
STATUS	PORTS		
NAMES			

0e98c993d74f	ollama/ollama	"/bin/ollama serve"	30 minutes ago
Up 30 minutes	0.0.0.0:11434->11434/tcp, :::11434->11434/tcp		
ollama			

Ollamaの実行

```
yo@Ryzen9:~$ docker exec -it ollama bash
root@0e98c993d74f:/# ls
bin boot dev etc home lib lib32 lib64 libx32 media mnt opt
proc root run sbin srv sys tmp usr var
```

モデルの確認

```
root@0e98c993d74f:/# ollama list
NAME          ID          SIZE   MODIFIED
gemma3n:e2b   719372f8c7de 5.6 GB  2 days ago
gemma3n:e4b   15cb39fd9394 7.5 GB  2 days ago
gemma3:4b     a2af6cc3eb7f 3.3 GB  2 days ago
gemma3:12b    f4031aab637d 8.1 GB  3 days ago
```

質問: ros noeticでollama APIの利用方法

```
root@0e98c993d74f:/# ollama run gemma3n:e4b
>>> ros noeticでロボットを製作しています。ollama APIに対して質問
をテキストで送信し、回答を得るためのプログラムをpythonで提示してく
ださい。
```

回答: Gemma3n:e4b

ROS Noetic で Ollama API を利用してテキスト質問に回答を得る
Python プログラム

このプログラムは、ROS Noetic 環境で Ollama API を利用してテキスト
質問を送信し、回答を得るための Python スクリプトです。

****必要なもの:****

- * ****Ollama:**** <https://ollama.com/> からダウンロードしてインストールしてください。
- * ****Python:**** 3.7 以上
- * ****ROS Noetic:**** ROS Noetic 環境がセットアップされていること。
- * ****`ollama` Python ライブラリ:**** `pip install ollama` でインストールします。

* ``rospy``: ROS Python 開発に必要なライブラリ。ROS 環境にインストールされているはずです。

****プログラム:****

```
``python
#!/usr/bin/env python

import rospy
import ollama
from std_msgs.msg import String

def ask_ollama(question):
    """
    Ollama API に質問を送信し、回答を得る関数。

    Args:
        question (str): 質問テキスト。

    Returns:
        str: Ollama からの回答。エラーが発生した場合は None を返す。
    """
    try:
        # Ollama API への接続
        response = ollama.Completion.create(
            model="llama2", # 使用するモデルを指定 (例: llama2)
            prompt=question,
            stream=False, # ストリーミング出力の有無
            temperature=0.7, # 温度パラメータ (創造性調整)
            top_p=0.95, # top_p パラメータ (多様性調整)
            # 他のパラメータも必要に応じて設定可能
        )

        # 回答の取得
        return response.choices[0].text.strip()

    except Exception as e:
        rospy.logerr(f'Ollama API 呼び出しエラー: {e}')
```

```
return None
```

```
def callback(data):
```

```
    """
```

ROS トピックからのメッセージを受け取り、Ollama に質問を送信し、
回答を publish する関数。

Args:

data (String): ROS トピックから受信したメッセージ。質問テキスト
が含まれていると想定。

```
    """
```

```
    question = data.data
```

```
    answer = ask_ollama(question)
```

```
    if answer:
```

```
        rospy.loginfo(f"質問: {question}")
```

```
        rospy.loginfo(f"回答: {answer}")
```

```
        # 回答を ROS トピックに publish
```

```
        answer_pub.publish(answer)
```

```
    else:
```

```
        rospy.logwarn(f'Ollama からの回答を取得できませんでした。質問: {question}')
```

```
if __name__ == '__main__':
```

```
    rospy.init_node('ollama_question_answerer', anonymous=True)
```

```
    # ROS トピックとメッセージの定義
```

```
    question_topic = '/question'
```

```
    answer_topic = '/answer'
```

```
    question_pub = rospy.Publisher(question_topic, String,  
    queue_size=10)
```

```
    answer_pub = rospy.Publisher(answer_topic, String,  
    queue_size=10)
```

```
    # トピックのサブスクリバ
```

```
    rospy.Subscriber(question_topic, String, callback)
```

```
rospy.spin()
...
```

****設定:****

1. ****モデルの指定:**** ``ollama.Completion.create()`` の ``model`` パラメータで、使用する Ollama モデルを指定します。
``llama2`` はデフォルトで利用可能なモデルですが、他のモデルも指定できます。Ollama で利用可能なモデルを確認するには、``ollama list`` コマンドを実行してください。
2. ****パラメータの調整:**** ``temperature``、``top_p`` などのパラメータを調整することで、回答の創造性や多様性を制御できます。
3. ****ROS トピックの定義:**** ``question_topic`` と ``answer_topic`` を、質問と回答を交換するための ROS トピック名に設定します。
4. ****Ollama の実行:**** Ollama を実行し、``ollama serve`` コマンドを実行します。

****使い方:****

1. 上記の Python スクリプトを保存します (例: ``ollama_node.py``)。
2. スクリプトを実行します。

```
```bash
roslaunch your_package_name ollama_node.py
```
```

(``your_package_name`` は、パッケージ名に置き換えてください)
3. 質問テキストを ``/question`` トピックに publish します。

```
```bash
rostopic pub /question std_msgs/String "data: 'あなたの質問テキスト'"
```
```
4. Ollama からの回答が ``/answer`` トピックに publish されます。

****注意点:****

* ****エラー処理:**** ``try...except`` ブロックでエラー処理を行っています
が、より詳細なエラーハンドリングを追加することを推奨

します。

* ****セキュリティ:**** Ollama API の認証が必要な場合は、API キーなどを適切に設定してください。

* ****リソース:**** Ollama はリソースを消費するため、実行環境に十分なリソースがあることを確認してください。

* ****モデルの互換性:**** 使用する Ollama モデルが、`ollama` Python ライブラリと互換性があることを確認してください。

* ****ストリーミング:**** `stream=True` に設定すると、Ollama からの回答をストリーミング形式で受け取ることができます。

****補足:****

* このプログラムは基本的な例であり、必要に応じて機能を拡張できます。

* 例えば、質問と回答の履歴を保存したり、特定のキーワードを含む質問をフィルタリングしたりすることができます。

* ROS のメッセージ型をカスタマイズすることで、より複雑なデータ構造を送信できます。

このプログラムが、ROS Noetic 環境で Ollama API を利用する上での助けになれば幸いです。

結論

gemma3:12bは、GPU無しで実行することができない。

上記を除いて、ダウンロードしたモデルのなかでサイズが最も大きいのは、gemma3n:e4bである。このモデルは実行可能である。よって、今回はe4bを使っていく。ここで、ollama runからモデルを使えるようになるまで、概ね20秒ほどかかる。起動時の自動化のとき、配慮する必要がある。

今後の予定

- ・生成AIをggmlからOllama に変更し、モデルをGemini3n:e4bとする。
- ・テキスト音声出力をpyopenjtalkからVOICEVOXに変更する。
- ・音声テキスト変換の辞書ファイルをsmallからnormalに変更する。
- ・顔を表示するシステムをqt5に変更する。
- ・顔を認識するシステムを構築する。

- ・移動装置を制作する。
- ・顔に向かって進む。
- ・認識した顔の方向に進むようにする。
- ・一定の場所に落ち着くようにする。
- ・認識した顔に、話しかけるようにする。

これまで通り、一度作成したシステムは、できるだけ残す。「おもしろパソコン」も可能な限り、そのまま残す。過去に作ったものは、今後、新しいものを作るときの貴重な資料となる。失敗したものは、そのまま残し、次に活かしていく。

参考

 20250511生成AI用PCの準備