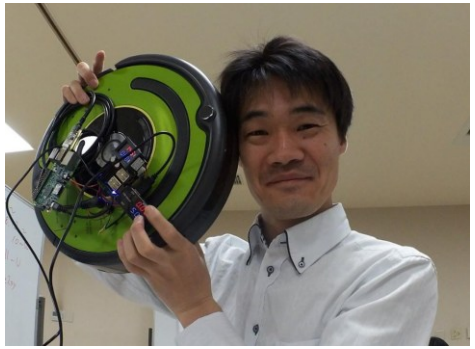
# 組込みでもLocalLLMしたい！

細合 晋太郎 （ものつくり大学）



ものつくり大学　技能工芸学部　情報メカトロニクス学科　講師
ほそあい　　しんたろう
# 細合 晋太郎

専門分野：ロボット，組込みシステム，ソフトウェア工学

ソフトウェア工学という道具を使って，組込みシステム（家電や自動車など様々なものに搭載されている小さなコンピュータ）やロボットの開発を支援する研究

略歴

2005年　近畿大学 学士卒
2007年　北陸先端科学技術大学院大学 修士卒
2012年　九州大学 学術研究員 (enPiT EMB)
2015年　北陸先端科学技術大学院大学 博士号取得
2016年　株式会社チェンジビジョン　正社員
2022年　東京大学　特任研究員
2024年　ものつくり大学　講師

## 「共創ロボティクス研究室」

共創(きょうそう)：異なる立場の人や団体が協力して，新たな価値を生み出す
共創ロボティクス：人・ロボットが協力して新たな価値を生み出す

ロボットと仲良くしようよ研究室って言ってます

沈むのがすきです.

# 最近 何してんの？

ものつくり大学でなんか作ってます！

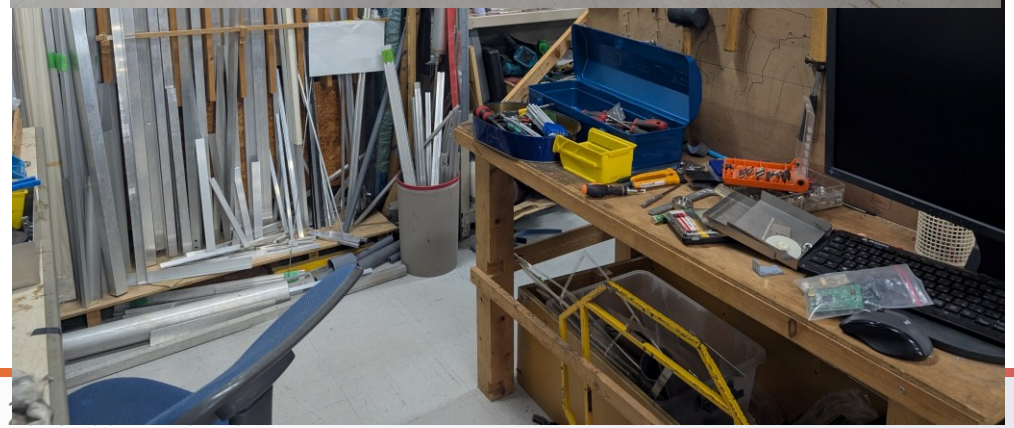
# 加工機一覧

- ○溶接
- ○旋盤・フライス盤
- ○5軸加工機
- ○ロボドリル
- ○ワイヤ放電
- ○射出成型機
- ○タレットパンチプレス
- ○シャーリング
- ○レーザーカッター
- ○NCベンダー



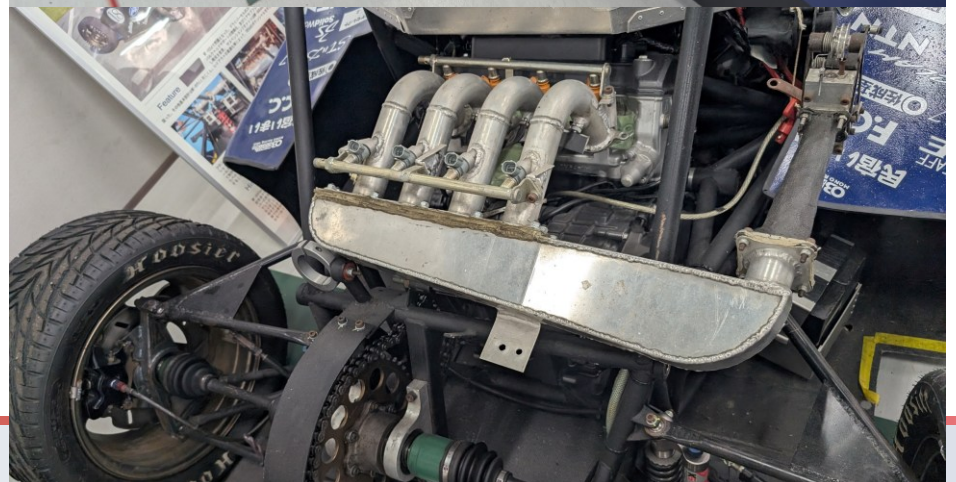



# ロボコンプロジェクト（NHKロボコン）

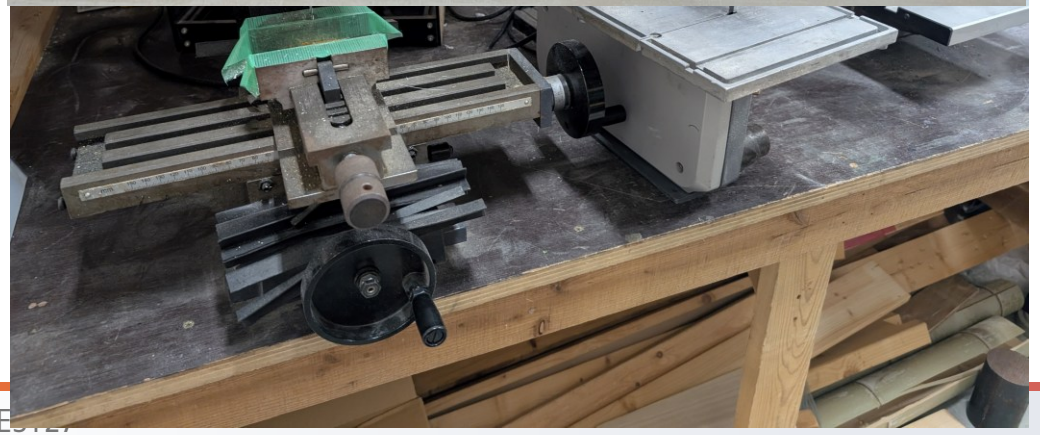


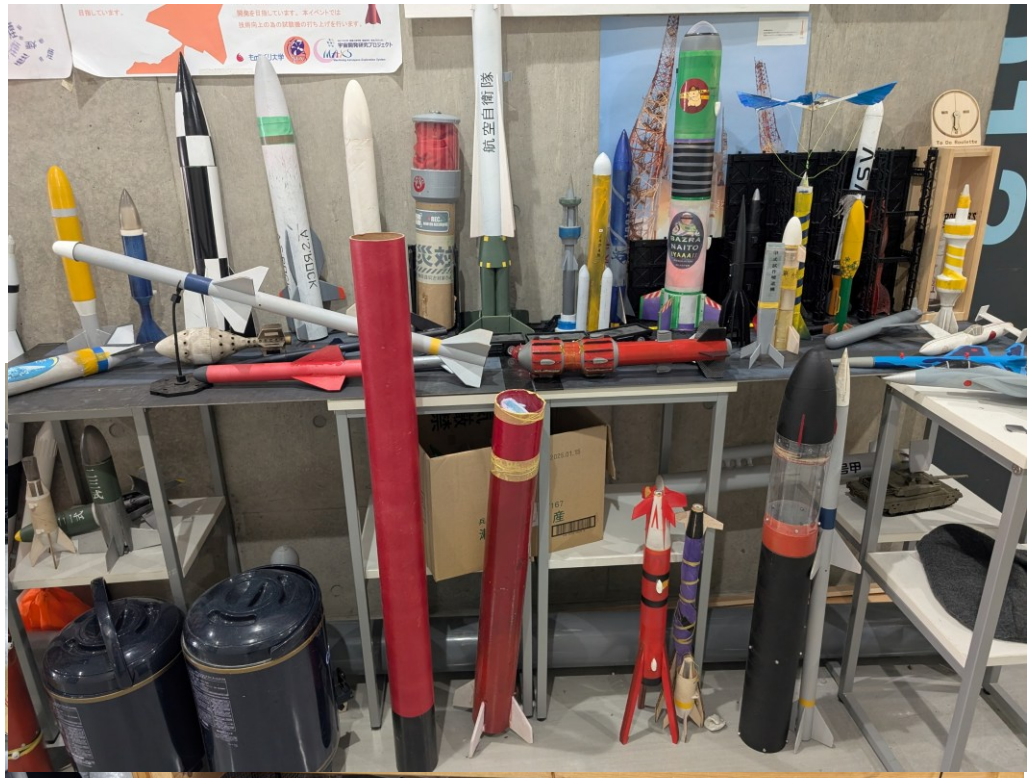
# 学生フォーミュラプロジェクト

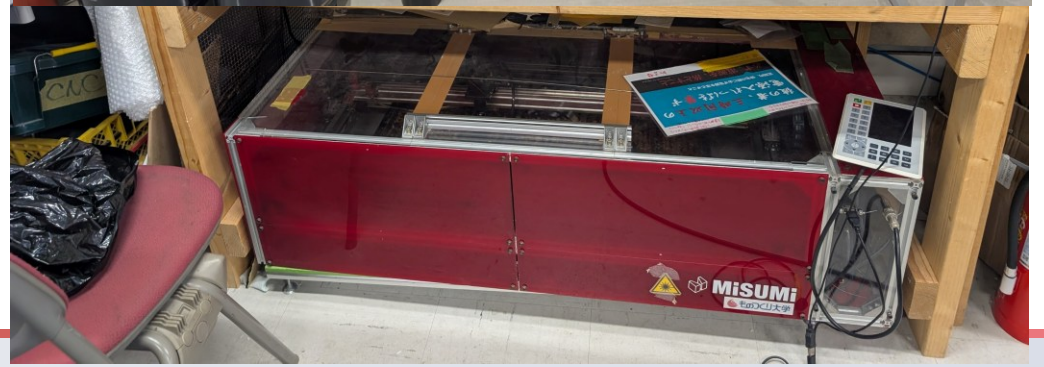




# 宇宙開発研究プロジェクト(種子島ロケットコンテスト)



10:35

# 今回のお品書き

70分枠
講義10分
演習40分
おまけ10分
バッファ・片付け10分

- 座学
  - 生成AI系の動向
  - Local LLM，Phi4
  - 今回準備したもの
- 演習
  - 生成AIと話してみよう（基本操作込みで5分）
  - ちょいチューニング：日本語，短文（3分）
  - Systemチューニング：日本語，短文（7分）
  - Function Call：テンプレと書き換え，Hello LLM（10分）
  - 組込みといえばLチカ！（5分)
  - LLM/FCからLチカ！（5分）
  - ボタンは押されてる？（5分）
  - ボタンを押したら（5分）
  - ボタンを押したら，ＬＬＭから問いかけて，ＬＥＤを付ける（時間いっぱいまで）
- Option（残り15分程度で）
  - 音声認識・音声合成，VOKS，Voicevoxのデモ
  - Local LLM と 組合せ．ボタンを押したら問いかけ，音声を認識してLEDを付ける
- おわりに（１分程度）
  - 今回用意したのは，先人が積み上げてくれたライブラリを組み合わせただけ
  - なんか自分で作れるような気がしてきますよね
  - 組込みをイゴかすのは我々組込みエンジニアが一番得意！面白いことしてSNSやイベントでドヤりましょう，成果はしっかり共有しましょう

# モチベーション

- お家に手軽にお話できるシステムを導入したい
- ご自宅にサーバラックを組んでる誤家庭やツヨツヨGPUマシンを誰しも持ってるとは限らないので，
  誰でも手軽に手を出せるRPiなどでLocal LLMを導入したい
- 折角RPiでやるなら，組込みっぽくGPIOも叩きたい
- 折角やるならマイクもスピーカーも繋いでお話したい

# 生成AIとは，ChatGPTとは，Local LLMとは

(ChatGPTでコンテンツ作って貼ろうとも思いましたが，あまりに不毛なので．．)

ご自身でChatGPTに聞いといてください

# Local LLMだと何がうれしいの？

- ネット環境がなくても，お金がなくても，無償で利用できる
- 自分で好きにモデルを再教育できる．機密情報でも気兼ねなく教えられる
- ローカル環境と強く結びついたシステムを作れる（・・と思ってたんですが，どう作ろうが実質あんまり変わりないです）

とはいえ，どう足掻いても通常のマシンじゃ本家には敵わないので，

ネット環境が豊富かつ，API料金を払う事を厭わないなら，OpenAI API等に頼ってしまうのが楽
（一方で，ほんとにフルスペックのLLMが要るのかい？ってタスクは割と多いような気もします．最高級のツヨツヨAIを使って，あなたがしたかった事って何ですか？

# Ollamaとは

- ローカルでLLMモデルを動かすための実行環境やモデル管理，ツール類を詰め込んだもの
- めちゃくちゃお手軽にローカルLLMを試すことができる
- キャラがKawaii

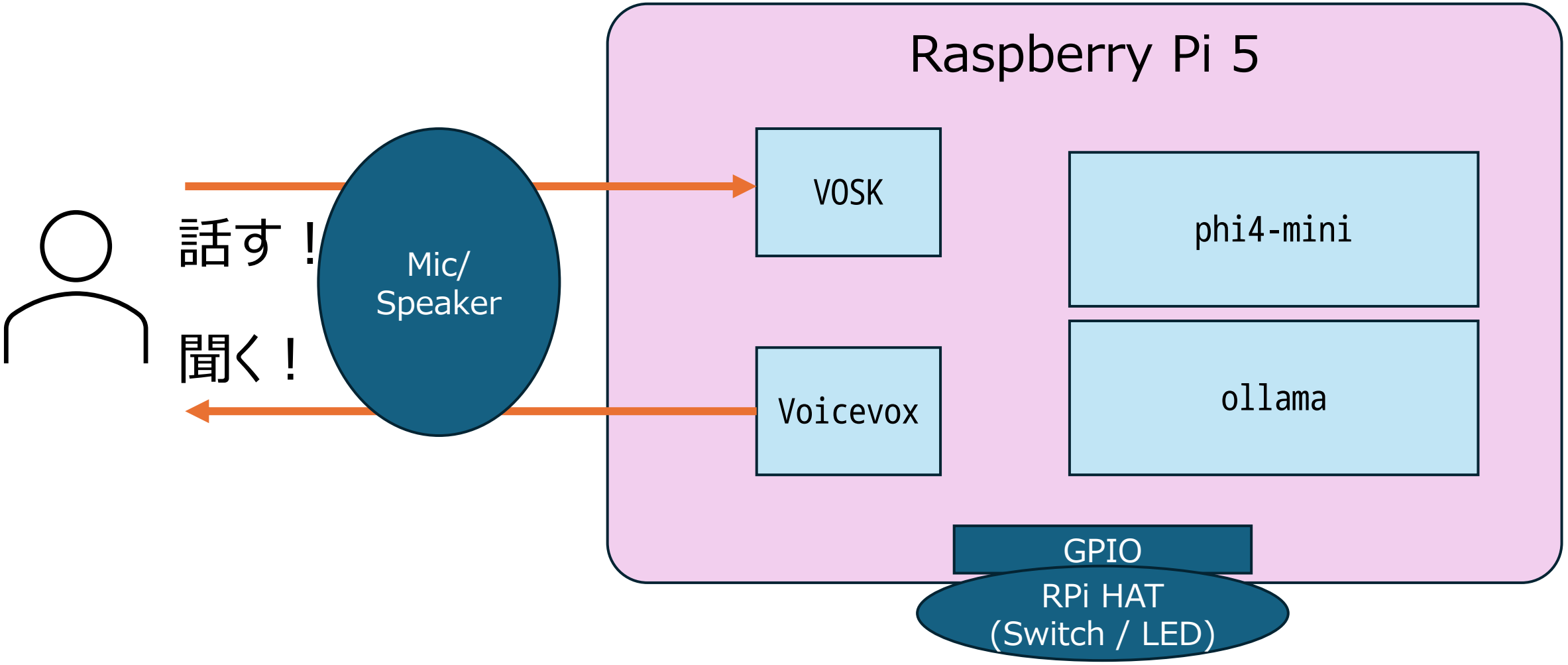
# Local LLMで使えるモデル比較

| モデルファミリー | 提供元（製造元） | 特徴 | 提供パラメータ規模（代表） |
|---|---|---|---|
| **Phi** | Microsoft | 小型高性能モデル。教育データ主体で学習。関数呼び出し・マルチリンガル対応。軽量でローカル適正高い。 | 3.8B（Mini）、14B（Medium, Phi-4） |
| **Mistral** | Mistral.ai | 軽量かつ高速。Grouped Query Attention採用。ベンチマークでも安定した高評価。 | 7B（Mistral）、12B（Nemo） |
| **Gemma** | Google DeepMind | Gemini技術ベース。安全性・公平性重視。サイズ展開が豊富でローカル利用にも適する。 | 2B、9B、27B |
| **DeepSeek** | DeepSeek AI | Mixture-of-Experts構造を採用。長文処理・推論性能に優れる。中国語や数理処理に強い。 | 7B、16B Lite、67B、236B、671B（MoE） |
| **LLaMA** | Meta（Facebook） | 標準的な大規模言語モデル。多用途に対応。Vision対応版やスケーラブルな構成も。 | 1B、3B、8B、70B、109B（Scout）、400B（Maverick） |

10:40

# 演習！

# 今回の構成


Raspberry Pi 5
話す！
聞く！
Mic/
Speaker
VOSK
Voicevox
phi4-mini
ollama
GPIO
RPi HAT
(Switch / LED)

# 凡例

濃い青の枠は，ターミナル入力を示します． 一行1コマンドとして1行毎に入力してください．

```
sudo apt update
sudo apt install tmux
```

コマンドが複数行にまたがってしまう場合は，{ で囲って示します．

緑色の背景は，ファイルの内容を表します．
また右肩のピンクの枠はファイル名です．

led_sample.py

```
import gpiod
import time
from gpiod.line import Direction, Value

LED_R = 17
LED_Y = 27
LED_G = 22

with gpiod.request_lines(
    "/dev/gpiochip4",
    consumer="blink-example",
    config={
      LED_R: gpiod.LineSettings(direction=Direction.OUTPUT, output_value=Value.INACTIVE),
```

# まずはログインとWiFi設定

Raspberry Piの画面でログインする

| | |
|---|---|
| User | ccrlab |
| Pass | ccrlabpass |

作成時とWiFiが異なってしまったため，変更をお願いします．

右上の  アイコンを選択し，以下のWiFiを選択する

| | |
|---|---|
| SSID | swest27c-a |
| Pass | suimeikan |

# LLMとご挨拶

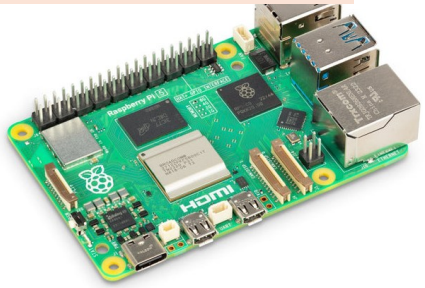
```
cd swest27
ollama run phi4-mini
```

>>>
（起動にしばらくかかる >>> まで表示されたら会話してみましょう）
※Ctrl + c で会話中で打ち切ることができます
※Ctrl + d でollamaを終了します

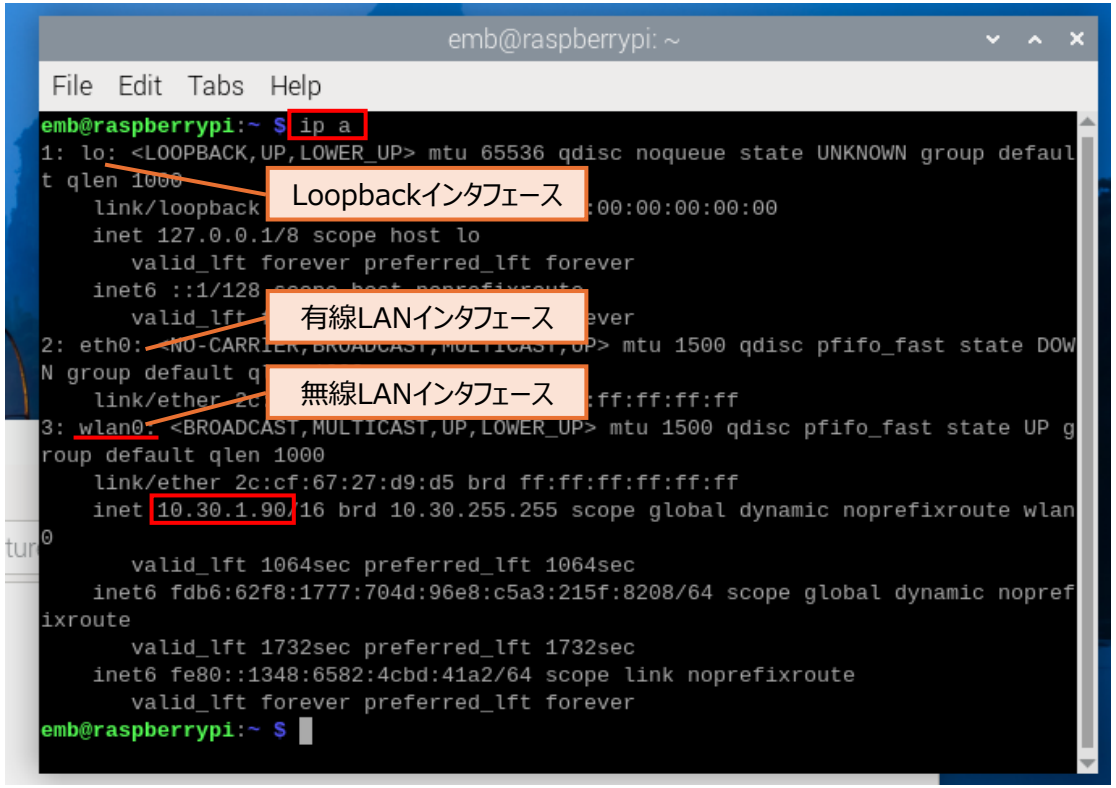
プログラムを手動で写経するには時間が足りないため，
SSHでコピペすることを強くお勧めします



# IPアドレスの確認



- まずは，「ip a」とコマンドを打ち，Raspberry PiのIPアドレスを確認する
- ip コマンドはネットワーク関連の様々な情報の表示や設定が行えるコマンド．今回はaオプションで，各ネットワークインタフェースの情報を表示している．
- 学内WiFiから取得したIPアドレスを調べるので，wlan0(無線LANインタフェース)のinetアドレスを確認する
- 学内では，大抵の場合
  10.30.**.**
  のアドレスが割り当てられる



# Windowsからのssh接続1

- Windowsでコマンドプロンプトを開く
  - 方法1：検索バーから，「コマンドプロンプト」または「cmd」で検索
  - 方法2：ファイルエクスプローラーを開き，何もないところで右クリック＞ターミナルで開く（Win11だけ？）
  - 方法3：Winキーを押してメニューを開き，cmdと打つ
  - 方法4：Winキー ＋ R で“ファイル名を指定して実行“を開き，cmdと打つ
- 開いたコマンドプロンプトで，以下のコマンドを入力

```
ssh ccrlab@00.00.00.00 （赤字の所は先ほど調べたIPアドレスを入力）
```

- 初回の接続時は，信頼できる接続先なのか問われるので，yesと入力する
- 続いてパスワードを聞かれるので，「ccrlabpass」と入力する．（パスワードは打ち込んでも何も表示されない．打ち込めたと思ったらEnter）

複数名でSSHから利用しても構いませんが，
ollamaを複数人で同時起動する多分アカン感じになります

# もうちょっとシステムっぽくしたい

- 組込みシステムにしてはえらい饒舌やん・・？

- もう少し簡潔に受け答えするようにプロンプトで指示してみましょう
- 「あなたは機器に搭載されたエージェントです．できるだけ簡潔に日本語で応対してください．」

- いくつかプロンプトのパターンを変えてお気に入りのエージェントに仕立て上げてください
- また

```
/set system "指示"
```

とすることで，システムプロンプトとして反映される・・はずなのですが，結構無視されます．．

# Ollamaのコマンド

- /? で一覧を確認できます.
- /set system "〜"
  システムプロンプトを追加します（追加してるのか？）
- /set verbose
  文章生成時に関する様々な情報を表示してくれます

# Ollamaの/api/chatと/api/generate

- 先ほどはツール上でChat形式で応対をしていました
- Ollamaは停止後もサービスとして動作しており，WebAPI経由でエージェント的な働きをしてくれます

- 試しに，curlで叩いてみましょう．

```
curl -s http://localhost:11434/api/chat -H "Content-Type:
application/json" -d '{
    "model": "phi4-mini",
    "messages": [
      {"role": "user", "content": "こんにちは！"}
    ], "stream": false}'
```

1コマンドとして入力する

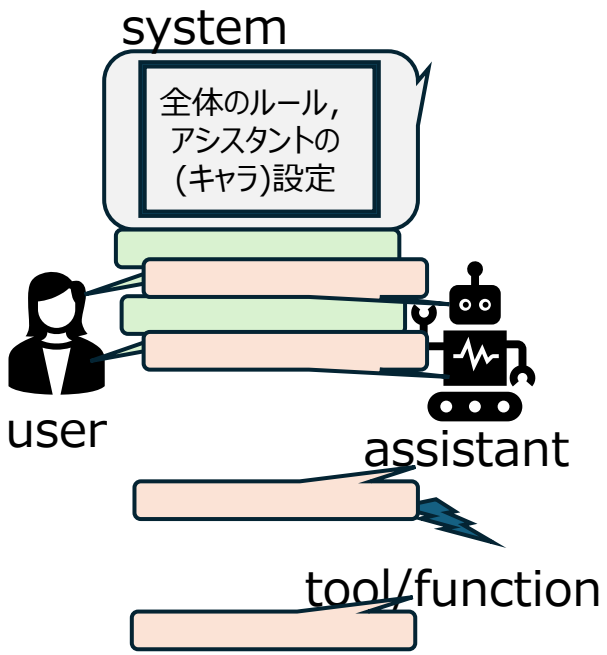
# Ollamaの/api/chatと/api/generate

| 特徴 | /api/generate | /api/chat |
| --- | --- | --- |
| 用途 | 単発のプロンプトと応答 | チャット履歴を維持して会話 |
| コンテキスト管理 | なし（明示的に含める必要あり） | 自動で過去履歴を保持 |
| 適している場面 | 単発の質問、Function Calling、テスト | 対話型アシスタント、会話の継続 |
| 登場バージョン | 初期から使用可能 | Ollama v0.1.34～対応 |
| 入力フォーマット | prompt に直接文字列を渡す | messages にリスト形式のロール付き会話履歴を渡す |

# Role

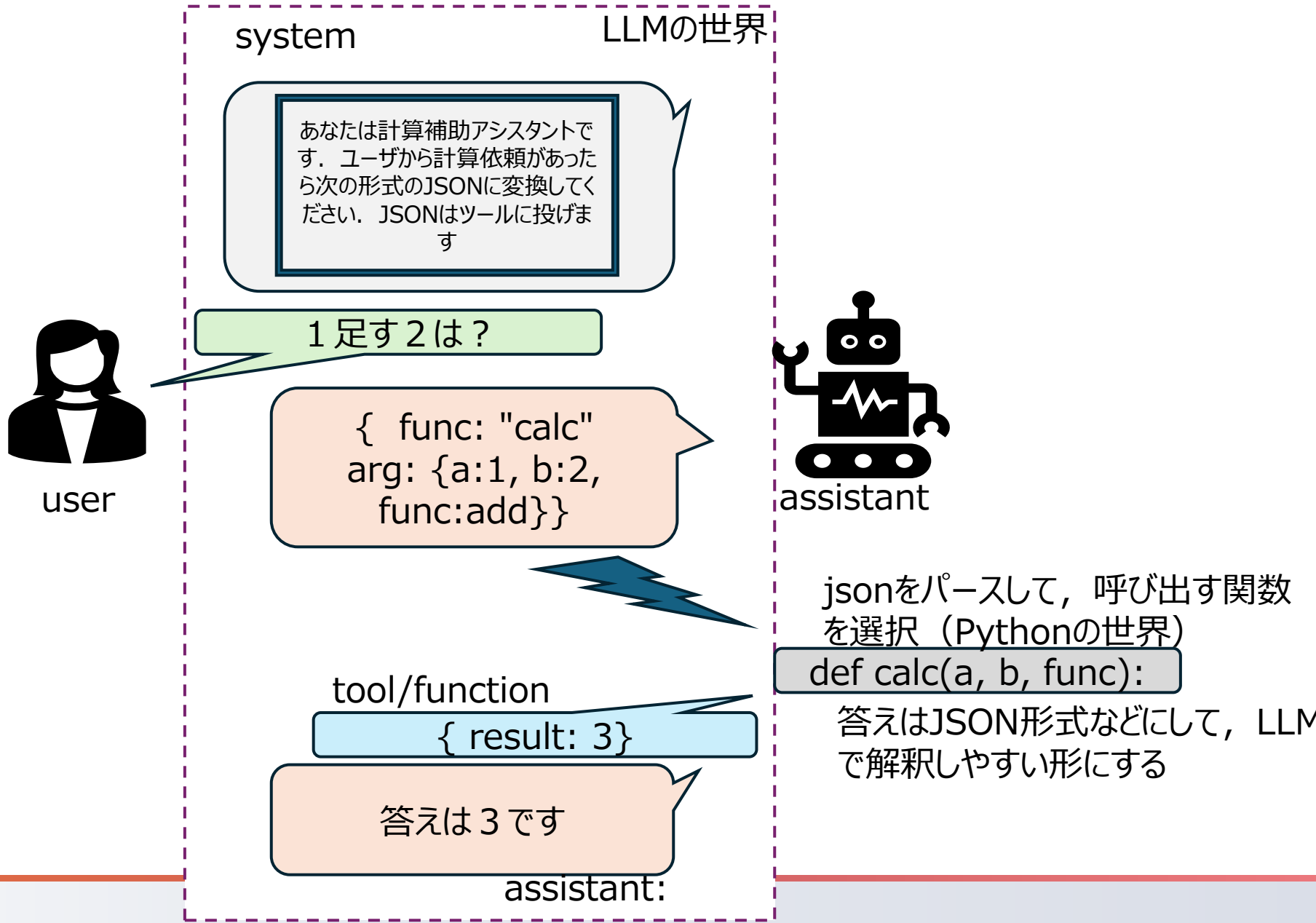
- 会話を構成する登場人物が決められている

| role | 説明 |
|---|---|
| system | モデル全体の振る舞いを定義。プロンプトの冒頭に指定されることが多い。 |
| user | ユーザーからの入力（通常のプロンプト）。 |
| assistant | LLM（アシスタント）が出力したメッセージ。 |
| tool | 関数呼び出し（function call）や、外部ツールの応答を示す。 |
| function | 呼び出された関数が返す応答を表す（tool と同じ意味として扱われることもあるが、OpenAI API互換を意識する場合に使用）。 |




```
curl -s http://localhost:11434/api/chat -H "Content-Type: application/json" -d '{ "model": "phi4-mini", "messages": [{"role": "system", "content": "あなたは日本語で応対するアシスタントです． 可能な限り簡潔に答えてください． "}, {"role": "user", "content": "組込みシステムでよく使われるCPUは何ですか？"} ], "stream": false}'
```

# Role


system
LLMの世界
あなたは計算補助アシスタントです．ユーザから計算依頼があったら次の形式のJSONに変換してください．JSONはツールに投げます
1 足す 2 は？
user
{ func: "calc"
arg: {a:1, b:2,
func:add}}
assistant
jsonをパースして，呼び出す関数を選択（Pythonの世界）
def calc(a, b, func):
tool/function
{ result: 3}
答えはJSON形式などにして，LLMで解釈しやすい形にする
答えは 3 です
assistant:

ChatGPTさんにFunction Callの例よろしくとお願いしたら，ちょっと違うものが帰ってきてしまった．が，ある意味分かりやすいかもなので，使います

# 無理くり関数呼び出しのソースコード

11:05

全体をコピーしてファイルを作成，実行(uv run calc.py)してください

calc.py

```
import requests
import json
import re

OLLAMA_URL = 'http://localhost:11434/api/generate'
MODEL = 'phi4-mini'
def calculate(a: float, b: float, op: str) -> float:
    if op == 'add':
        return a + b
    elif op == 'subtract':
        return a - b
    elif op == 'multiply':
        return a * b
    elif op == 'divide':
        return a / b
    else:
        raise ValueError("Unknown operation")

system_prompt = """あなたはユーザーの自然な要求から、適切な関数呼び出しを提案するAIです。
呼び出すべき関数：
{
  "function": "calculate",
  "parameters": {
    "a": float,
    "b": float,
    "op": "add | subtract | multiply | divide"
  }
}
返答は次のようなJSON形式で行ってください．引用形式でなく生のjsonを出力してください：
{
  "function_call": {
    "name": "calculate",
    "arguments": {
      "a": 12,
      "b": 3,
      "op": "divide"
    }
  }
}
"""
# ユーザー入力
user_prompt = "１０かける３は？"

# モデルへ送信
response = requests.post(OLLAMA_URL, json={
    "model": MODEL,
    "prompt": f"<|system|>¥n{system_prompt}¥n<|user|>¥n{user_prompt}¥n<|assistant|>",
    "stream": False
})

# モデル出力取得
raw_output = response.json()["response"]
print(raw_output)
# JSON風の出力を整形・読み込み
try:
    cleaned = raw_output.strip(`` ¥n')  # 先頭/末尾のバッククォート・空白除去
    parsed = json.loads(cleaned)
    if "function_call" in parsed:
        args = parsed["function_call"]["arguments"]
        result = calculate(**args)
        print(f"💡 計算結果: {result}")
    else:
        print("⚠ 関数呼び出しが見つかりませんでした")
except Exception as e:
    print("⚠ JSON解析エラー:", e)
```

# 無理くり関数呼び出しのソースコード　【解説】

```
import requests
import json
import re

OLLAMA_URL = 'http://localhost:11434/api/generate'
MODEL = 'phi4-mini'
def calculate(a: float, b: float, op: str) -> float:
    if op == 'add':
        return a + b
    elif op == 'subtract':
        return a - b
    elif op == 'multiply':
        return a * b
    elif op == 'divide':
        return a / b
    else:
        raise ValueError("Unknown operation")
```

LLMから呼んでほしい関数の本体

```
system_prompt = """あなたはユーザーの自然な要求から、適切な関数呼び出しを提案するAIです。
呼び出すべき関数：
{
  "function": "calculate",
  "parameters": {
    "a": float,
    "b": float,
    "op": "add | subtract | multiply | divide"
  }
}
返答は次のようなJSON形式で行ってください．引用形式でなく生のjsonを出力してください：
{
  "function_call": {
    "name": "calculate",
    "arguments": {
      "a": 12,
      "b": 3,
      "op": "divide"
    }
  }
}
"""
```

システムプロンプトここまで

```
# ユーザー入力
user_prompt = "１０かける３は？"
```

今回は入力は決め打ち

システムプロンプト．状況に応じて関数を呼び出すように誘導する

関数を正しく呼び出すために，JSONの定義を明確に伝える

```
# モデルへ送信
response = requests.post(OLLAMA_URL, json={
    "model": MODEL,
    "prompt": f"<|system|>¥n{system_prompt}¥n<|user|>¥n{user_prompt}¥n<|assistant|>",
    "stream": False
})
```

system，userをそれぞれ入力して，URLへ呼び出し

```
# モデル出力取得
raw_output = response.json()["response"]
print(raw_output)
# JSON風の出力を整形・読み込み
try:
    cleaned = raw_output.strip('` ¥n')  # 先頭/末尾のバッククォート・空白除去
    parsed = json.loads(cleaned)
    if "function_call" in parsed:
        args = parsed["function_call"]["arguments"]
        result = calculate(**args)
        print(f"💡 計算結果: {result}")
    else:
        print("⚠ 関数呼び出しが見つかりませんでした")
except Exception as e:
    print("⚠ JSON解析エラー:", e)
```

無理やりしゃべらせたJSONをパースして，マッチするなら，該当の関数を呼び出す．

# 組込みならLチカ！

- （RPi持ち込みの方申し訳ない・・，心のGPIOを灯してください）
- 準備

```
sudo apt install gpiod
uv add gpiod
```

- とりあえず光らせる

赤
黄
緑

```
gpioset gpiochip4 17=1
gpioset gpiochip4 27=1
gpioset gpiochip4 22=1

gpioset gpiochip4 17=0
gpioset gpiochip4 27=0
gpioset gpiochip4 22=0
```

- Pythonからならー＞

led_sample.py

```
import gpiod
import time
from gpiod.line import Direction, Value

LED_R = 17
LED_Y = 27
LED_G = 22

with gpiod.request_lines(
    "/dev/gpiochip4",
    consumer="blink-example",
    config={
      LED_R: gpiod.LineSettings(direction=Direction.OUTPUT, output_value=Value.INACTIVE),
      LED_Y: gpiod.LineSettings(direction=Direction.OUTPUT, output_value=Value.INACTIVE),
      LED_G: gpiod.LineSettings(direction=Direction.OUTPUT, output_value=Value.INACTIVE)},
) as request:
    request.set_value(LED_R, Value.ACTIVE)
    time.sleep(1)
    request.set_value(LED_Y, Value.ACTIVE)
    time.sleep(1)
    request.set_value(LED_G, Value.ACTIVE)
    time.sleep(1)
    request.set_value(LED_G, Value.INACTIVE)
    time.sleep(1)
    request.set_value(LED_Y, Value.INACTIVE)
    time.sleep(1)
    request.set_value(LED_R, Value.INACTIVE)
    time.sleep(1)
```

```
uv run led_sample.py
```

INSTITUTE OF TECHNOLOGISTS

11:20

# LLMからLチカ！！

全体をコピーしてファイルを作成，実行してください

llm_led.py

```
import requests
import json
import re
import gpiod
from gpiod.line import Direction, Value

OLLAMA_URL = 'http://localhost:11434/api/generate'
MODEL = 'phi4-mini'  # または 'phi3.5' など

LED_R = 17
LED_Y = 27
LED_G = 22

# 実行可能な関数定義
def parse_value(str):
    if str == "Value.ACTIVE" :
        return Value.ACTIVE
    else:
        return Value.INACTIVE

def led(led_r: str, led_y: str, led_g: str) -> bool:
    r_value = parse_value(led_r)
    y_value = parse_value(led_y)
    g_value = parse_value(led_g)
    with gpiod.request_lines(
        "/dev/gpiochip4",
        consumer="blink-example",
        config={
            LED_R: gpiod.LineSettings(direction=Direction.OUTPUT, output_value=Value.INACTIVE),
            LED_Y: gpiod.LineSettings(direction=Direction.OUTPUT, output_value=Value.INACTIVE),
            LED_G: gpiod.LineSettings(direction=Direction.OUTPUT, output_value=Value.INACTIVE)},
    ) as request:
            request.set_value(LED_R, r_value)
            request.set_value(LED_Y, y_value)
            request.set_value(LED_G, g_value)

# プロンプト（関数を使うよう誘導）
system_prompt = """
あなたはユーザーの自然な要求から、3つのLEDのオンオフを切り替える関数の呼び出しを提案するAIです。
led_rは赤、led_yは黄、led_gは緑です．付けてと言われた時はValue.ACTIVE，消してと言われた時はValue.INACTIVEにしてください．
言及されなかった色はValue.INACTIVEとしてください．
呼び出すべき関数：
{
  "function": "led",
  "parameters": {
    "led_r": "Value.ACTIVE | Value.INACTIVE"
    "led_y": "Value.ACTIVE | Value.INACTIVE"
    "led_g": "Value.ACTIVE | Value.INACTIVE"
  }
}
返答は次のようなJSON形式で行ってください．引用形式でなく生のjsonを出力してください：
{
  "function_call": {
    "name": "led",
    "arguments": {
      "led_r": "Value.ACTIVE",
 "led_y": "Value.INACTIVE",
      "led_g": "Value.ACTIVE"
    }
  }
}
"""

# ユーザー入力
user_prompt = "黄と赤をつけて"

# モデルへ送信
response = requests.post(OLLAMA_URL, json={
    "model": MODEL,
    "prompt": f"<|system|>¥n{system_prompt}¥n<|user|>¥n{user_prompt}¥n<|assistant|>",
    "stream": False
})

# モデル出力取得
raw_output = response.json()["response"]
print(raw_output)
# JSON風の出力を整形・読み込み
try:
    cleaned = raw_output.strip(`` ¥n')  # 先頭/末尾のバッククォート・空白除去
    parsed = json.loads(cleaned)
    if "function_call" in parsed:
        args = parsed["function_call"]["arguments"]
        result = led(**args)
        print(f"💡 計算結果: {result}")
    else:
        print("⚠ 関数呼び出しが見つかりませんでした")
except Exception as e:
    print("⚠ JSON解析エラー:", e)
```

ユーザプロンプトを書き換えて，
いろいろ試してみてください

11:30

# 音声認識したい https://alphacephei.com/vosk/

- VOSKがとても便利でした．当初Whisper系のものを利用しようとしてましたが，あれこれ面倒で・・．
- VOSKはOSS，Cross Platformな音声認識エンジンです．
- 多くの言語に対応したライブラリが用意されています
- 音声ファイルから認識する場合は以下を

```
vosk-transcriber -l ja -i output.wav
```

- 音声デバイスからの入力は以下のサンプルを参照してください

https://github.com/alphacep/vosk-api/blob/master/python/example/test_microphone.py

```
wget https://github.com/alphacep/vosk-api/blob/master/python/example/test_microphone.py

uv run test_microphone.py
```

# 音声合成もしたい

- VoicevoxはOSSの音声合成ライブラリです．多数の音声モデルが用意されており，手軽に音声合成を楽しむことが可能です．
- その他の音声合成は追えてません

- ↓のrun.pyのキモの部分はそれほど大きくないので，出力を音声合成し，再生させることはそれほど難しくないはず・・．

```
wget https://raw.githubusercontent.com/VOICEVOX/voicevox_core/refs/heads/main/example/python/run.py

uv run run.py ./models/0.vvm --style-id 3 --dict-dir ./voicevox_core/dict/open_jtalk_dic_utf_8-1.11
--onnxruntime ./voicevox_core/onnxruntime/lib/libvoicevox_onnxruntime.so.1.17.3 --text "こんにちわ"

play -t wav output.wav
```

# 音声でお願いして，音声で答えて欲しい

- 大変申し訳ない．．時間が足りませんでした．

- 一定区間でVOSKで聞き続け，認識したらUser Promptに食わせ，結果をLチカやその他デバイスに出しつつ，程よいテキストも一緒に返す．ようにすれば，できる気がしてるんです．できる気が・・

- また，今回は単発で動かしてますが，固有のChat IDを使いまわすことで，一連の流れや，別プロセスからのアクセスでも同一のユーザとの対話と見なすことができます．ので，ボタン監視や割込みのタイミングでAssistantプロンプトを叩いたり，時報毎に温度を測ってお知らせする，みたいな事も十分できそうですよね

# おわりに

11:37

- なんとなく組込みでLLMっぽい雰囲気になったでしょうか？
- まだレスポンスがだいぶ遅く，リアルタイム(人の許容待ち時間）にはまだまだですが，使い方次第では十分に実用に足るところまで近づいてきているように思います．
- 予算を足してJetson Orin Nano Superくらいまでスペックアップすれば，組込みサイズで十分リアルタイム対話できそうな可能性を感じます．
- また，すべてローカルでやる必要もなく，重い所だけ別PCやクラウドにお任せしてしまうのもありでしょう．

- ぜひ自分だけのAIアシスタントを作ってみてください

# 環境構築

- とりあえず操作方法だけ並べてます.
  導入したツール自体の説明はセッション内でします（するはず）

# 使用機材

- Raspberry Pi 5, RPi5用電源
- RPi用ディスプレイ
- キーボード，マウス
- microSD 32GB以上（64GB推奨）
- (opt)スピーカー
- (opt)マイク

# 使用環境

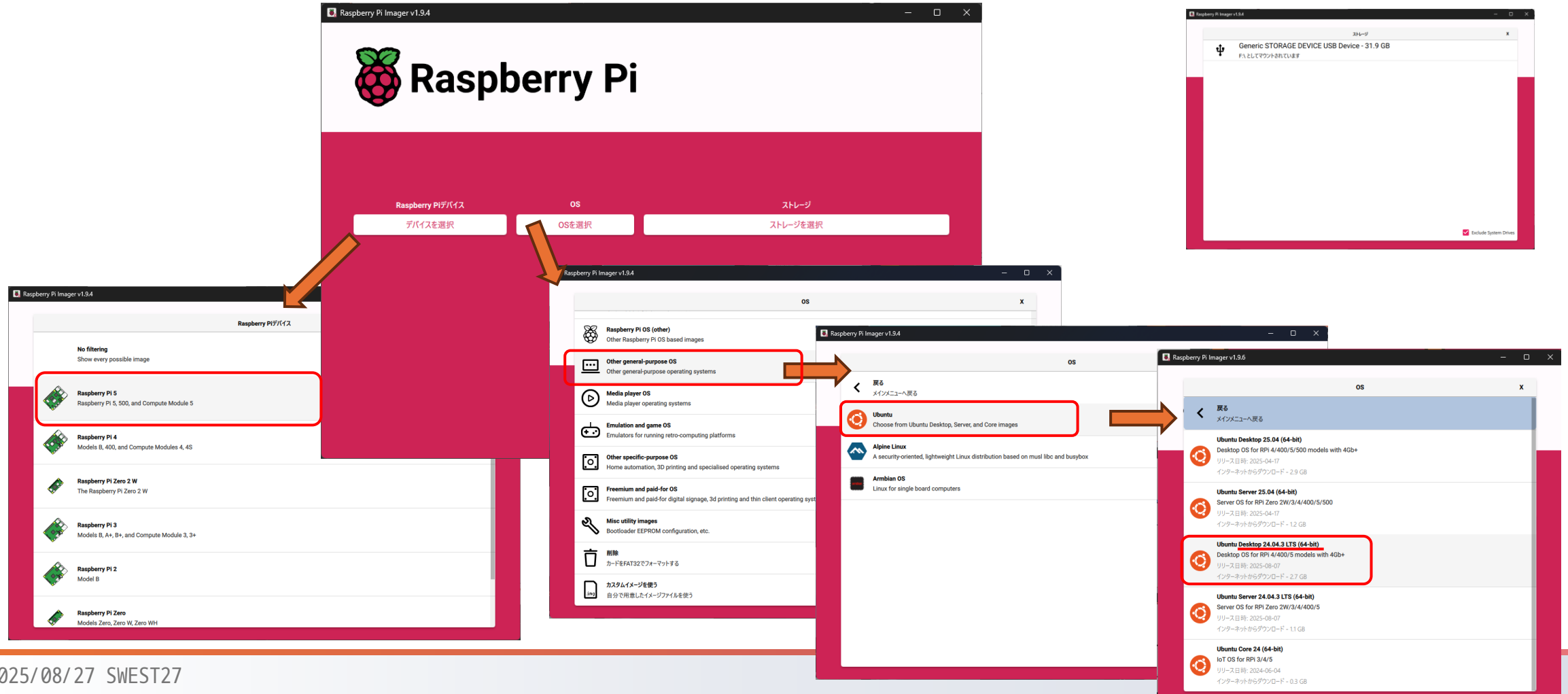
- Ubuntu Server 24.04.3 LTS(64-bit) + UbuntuMATE
  - バージョンは安定なLTSで
  - Raspberry Pi OSは若干使い勝手が苦手，AI周りのライブラリがUbuntu前提が多そうなので，RPiOSとの差異で悩みたくない
  - とはいえ，フルUbuntuは若干重い

  ということで，一旦Serverで入れて軽量デスクトップマネージャーを入れる方針

  - なんでUbuntuMATE？
    - なんかネットで軽いって言ってる人が使ってみた
    - そのほか
      - Lubuntu：あんまり使ったことがない．軽いらしい
      - Xfce：シンプルで軽い．ネズミさんがかわいい．

# Raspberry Pi イメージの作成

- 以下よりアプリをDL/インストール
  https://www.raspberrypi.com/software/



# Ubuntu 初期設定

- System Configuration
  言語：日本語　（英語の方が得意な方はお好きに）
- キーボード：Japanese - Japanese　（英字配列の人は適宜変更）
- 無線：ご自宅のWifi情報を入力
- 国：Tokyo
- あなたの情報を～：お好きに入力してください．パスワードを忘れないようにしてください．
  一旦，「自動的にログインする」にはしないでください．

しばし待ちます

# 初期設定

- 起動したUbuntuで，初期設定が出ますがNext/Skipで構いません．またアップデートも以降で行うのでダイアログが出てもスキップで．

- ターミナルを開いて(Ctrl+Alt+T)，以下を入力します．

```
sudo apt update
sudo apt upgrade
sudo apt install curl git p7zip build-essential ffmpeg sox openssh-server
```

（以降，青黒背景↑はUbuntuのシェル上での操作を表します．
一行ずつ入力し，エンターを押してください．複数行に渡るコマンドの場合は { で指示します．）

- ssh等が利用できる方は，sshでアクセスして操作しても構いませんが，その分セキュリティが弱くなります．十分注意して運用してください．

# 軽量化

以降，緑背景はファイルの中身を表します→

- UbuntuMATE
- Swap最適化
- GPUメモリ設定
- 軽量なアプリ類．等々
- →は，ChatGPTが作成してくれた
  これらを一気にやってくれるスクリプト．

```
nano setup.sh
```

- でファイルを開き，→をコピペして保存し，
  以下で実行

```
sudo bash ./setup.sh
```

- Reboot後，ログイン時に設定⚙から
  UbuntuでなくMateを選択する

```
#!/usr/bin/env bash
set -euo pipefail

### settings (必要に応じて調整)
SWAPSIZE_MB=2048        # スワップサイズ(MB)
GPU_MEM_MB=256          # GPUメモリ割当(MB)
FULL_UBUNTU_MATE=false  # true にすると ubuntu-mate-desktop を入れる（アプリ一式）

### checks
if [[ $EUID -ne 0 ]]; then
  echo "Please run as root: sudo $0"
  exit 1
fi

echo "==> 1) apt update/upgrade"
apt update
apt -y upgrade

echo "==> 2) Install MATE session"
if [[ "$FULL_UBUNTU_MATE" == "true" ]]; then
  apt -y install ubuntu-mate-desktop
else
  apt -y install mate-desktop-environment mate-desktop-environment-extras
fi

echo "==> 3) Useful/lightweight apps"
apt -y install chromium-browser pluma caja mate-terminal atril engrampa mate-tweak

echo "==> 4) Disable heavy GNOME Tracker services (if present)"
for SVC in tracker-miner-fs-3.service tracker-miner-rss-3.service tracker-miner-apps-3.service tracker-store-3.service; do
  systemctl disable --now "$SVC" 2>/dev/null || true
done

echo "==> 5) Increase swap (dphys-swapfile)"
apt -y install dphys-swapfile
swapoff -a || true
cp -a /etc/dphys-swapfile /etc/dphys-swapfile.bak.$(date +%F-%H%M%S) || true
sed -i "s/^#¥?CONF_SWAPSIZE=.*/CONF_SWAPSIZE=${SWAPSIZE_MB}/" /etc/dphys-swapfile || true
if ! grep -q "^CONF_SWAPSIZE=" /etc/dphys-swapfile; then
  echo "CONF_SWAPSIZE=${SWAPSIZE_MB}" >> /etc/dphys-swapfile
fi
dphys-swapfile setup
dphys-swapfile swapon

echo "==> 6) Set GPU memory in /boot/firmware/config.txt"
FWCFG="/boot/firmware/config.txt"
cp -a "$FWCFG" "${FWCFG}.bak.$(date +%F-%H%M%S)" || true
if grep -q "^gpu_mem=" "$FWCFG"; then
  sed -i "s/^gpu_mem=.*/gpu_mem=${GPU_MEM_MB}/" "$FWCFG"
else
  echo -e "¥n# Added by setup-mate-rpi5¥ngpu_mem=${GPU_MEM_MB}" >> "$FWCFG"
fi

echo "==> 7) Make MATE lighter via system-wide dconf (no compositor, less eye-candy)"
# 作成: /etc/dconf/db/local.d/99-mate-tweaks
mkdir -p /etc/dconf/db/local.d /etc/dconf/db/local.d/locks
cat >/etc/dconf/db/local.d/99-mate-tweaks <<'EOF'
[org/mate/marco/general]
compositing-manager=false

[org/mate/desktop/interface]
enable-animations=false
EOF

# 強制ロック（ユーザが後でGUIで変更したい場合はコメントアウトしてOK）
cat >/etc/dconf/db/local.d/locks/mate-tweaks <<'EOF'
/org/mate/marco/general/compositing-manager
/org/mate/desktop/interface/enable-animations
EOF

dconf update || true

echo "==> 8) Keep gdm3 (default) as display manager (no change)"
# 何もしません。gdm3のままでOK

echo "==> 9) Set default boot target to graphical (just in case)"
systemctl set-default graphical.target

echo "==> 10) Done. Reboot recommended."
echo "----------------------------------------"
echo "Setup finished. Please reboot:"
echo "  sudo reboot"
echo "Reboot後、ログイン画面の歯車アイコンから MATE を選択してください。"
```

# uv

- Pythonのパッケージ管理があまりにも****なので，最近流行りらしいuvを入れます．
- 次のコマンドでインストールできます．

```
curl -LsSf https://astral.sh/uv/install.sh | sh

 [インストール後，指示通り以下を実行]
source $HOME/.local/bin/env
```

- また以降の作業は，swest27ディレクトリを作業ディレクトリとして話を進めます．

```
cd
mkdir swest27
cd swest27
uv init
```

# Local LLMの導入

- ここからの作業は結構な量のダウンロードが発生します．
  ネットワーク環境の良いところで作業しましょう．

# Ollamaのインストール

```
curl -fsSL https://ollama.com/install.sh | sh
```

- かなり時間が掛かります

（このところサーバが混みあっているのか，途中でエラーが出ることもあります．比較的混んでいなさそうな時間を狙って再度トライ）

- モデルの導入と実行

```
ollama run phi4-mini
```

こちらも時間が掛かります．
モデルのDLが終わると，Local LLMとおしゃべりできます．（相当重いです）なお，日本語で答えてくださいとお願いしておけば，ちゃんと日本語でもしゃべってくれます．

# Speech to Text / Text to Speech

# スピーカー・マイクの設定

- Raspberry Piに外部スピーカー，特にUSBスピーカー・マイクを接続した際は，優先順位が異なるため設定を変更する必要がある．

```
lsusb
  [ USBデバイスの一覧が表示される ]
cat /proc/asound/cards
  [ カード名が表示される ]
cat /proc/asound/[CARDNAME]/pcm0p/info
  [ カード詳細が表示される ]
sudo nano /usr/share/alsa/alsa.conf
  以下の内容の箇所を探し，数値の部分を上記で見つけたカードの番号を入力

    defaults.ctl.card 2
    defaults.pcm.card 2
    defaults.pcm.device 0

sudo vi /etc/modprobe.d/alsa-base.conf
  以下の内容の箇所を探し，数値の部分を上記で見つけたカードの番号を入力

    options snd_usb_audio index=0

sudo gpasswd -a $USER audio
sudo reboot
```

上記で見つけたカード名を入力．asound/まで打ってTabで補完を出すと分かりやすい

音声デバイスの権限が得れてないことがあるので，自身をグループに追加しておく

設定反映のため，再起動する

# マイク・スピーカーのテスト

```
・マイクのテスト
sox -d -r 44100 -b 16 output.wav
  [マイクに向かって歌う．気が済んだらCtrl＋C で終了]

・スピーカーのテスト
play -t wav output.wav
  [♪～ 美しい歌声 ～♪]
```

# Voicevox Python Lib

一行で
入力する

以降も
{が
付いている
箇所はす
べて一行
で

```
cd ~/swest27/
uv add
https://github.com/VOICEVOX/voicevox_core/releases/download/0.16.0/voicevox_core-0.16.0-cp310-abi3-manylinux_2_34_aarch64.whl

wget https://github.com/VOICEVOX/voicevox_core/releases/download/0.16.0/download-linux-arm64

chmod +x download-linux-arm64
```

どちらか
2択です

```
mkdir -p models
curl -L -o models/0.vvm
https://raw.githubusercontent.com/VOICEVOX/voicevox_vvm/main/vvms/0.vvm
./download-linux-arm64 --exclude models
```

音声を1種類だけ入れる場合

```
./download-linux-arm64
```

全部入れる場合（1GB程度になったはず）



いずれの場合も同意確認があるので，内容を確認しq，yを押してください

# Voicevoxの動作確認

```
wget
https://raw.githubusercontent.com/VOICEVOX/voicevox_core/refs/heads/main/example/python/run.py


uv run run.py ./models/0.vvm --style-id 3 --dict-
dir ./voicevox_core/dict/open_jtalk_dic_utf_8-1.11 --
onnxruntime ./voicevox_core/onnxruntime/lib/libvoicevox_onnxruntime.so.1.17.3 --text "こんにちわ"


play -t wav output.wav
```

# VOSKの導入と動作確認

```
uv tool install vosk

vosk-transcriber -l ja -i output.wav
```

←先ほどVoicevoxで生成したもの
soxで音声を録音したものを使ってみてもよい