

# GenAI at the Edge Laboratory

---

Prof. Marcelo J. Rovai

[rovai@unifei.edu.br](mailto:rovai@unifei.edu.br)

UNIFEI - Federal University of Itajuba, Brazil

TinyML4D - Academic Network Co-Chair

EdgeAIP - Academia-Industry Partnership Co-Chair



**LLM / SLM**

Large Language Model / Small Language Models



# Neural Network Architectures

Vibration  
Analysis

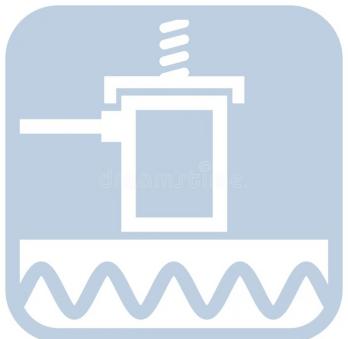


Image  
Classification



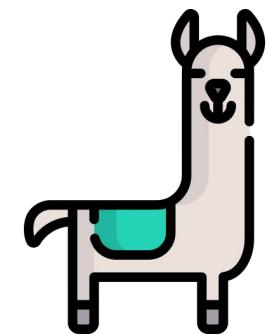
Text  
Generation



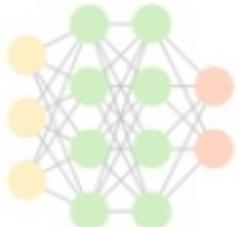
Image  
Generation



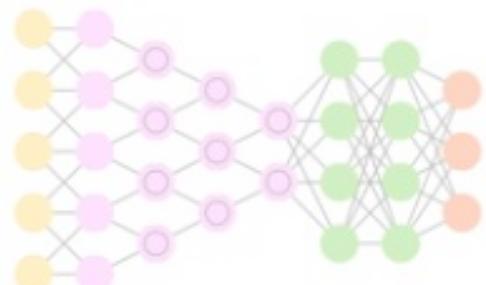
Large Language  
Models- LLMs



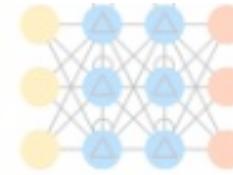
MLP - Deep Neural Network



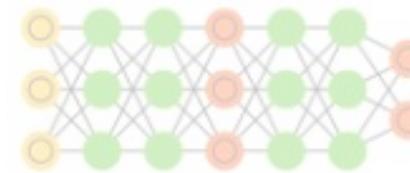
CNN - Convolutional NN



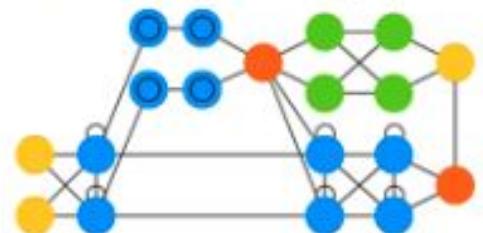
RNN - Recurrent NN (GRU/LSTM)



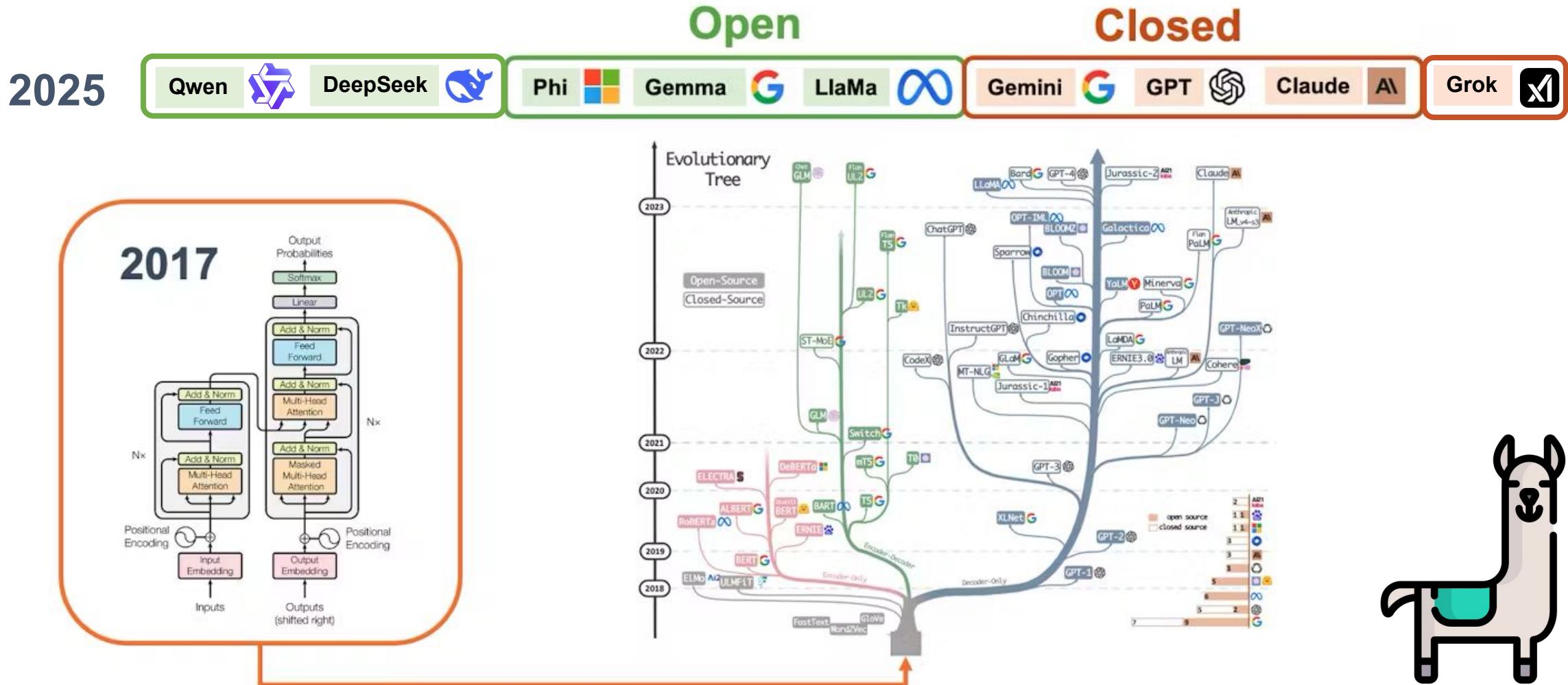
GAN - Generative Adversarial N.



AN - Attention (Transformers)



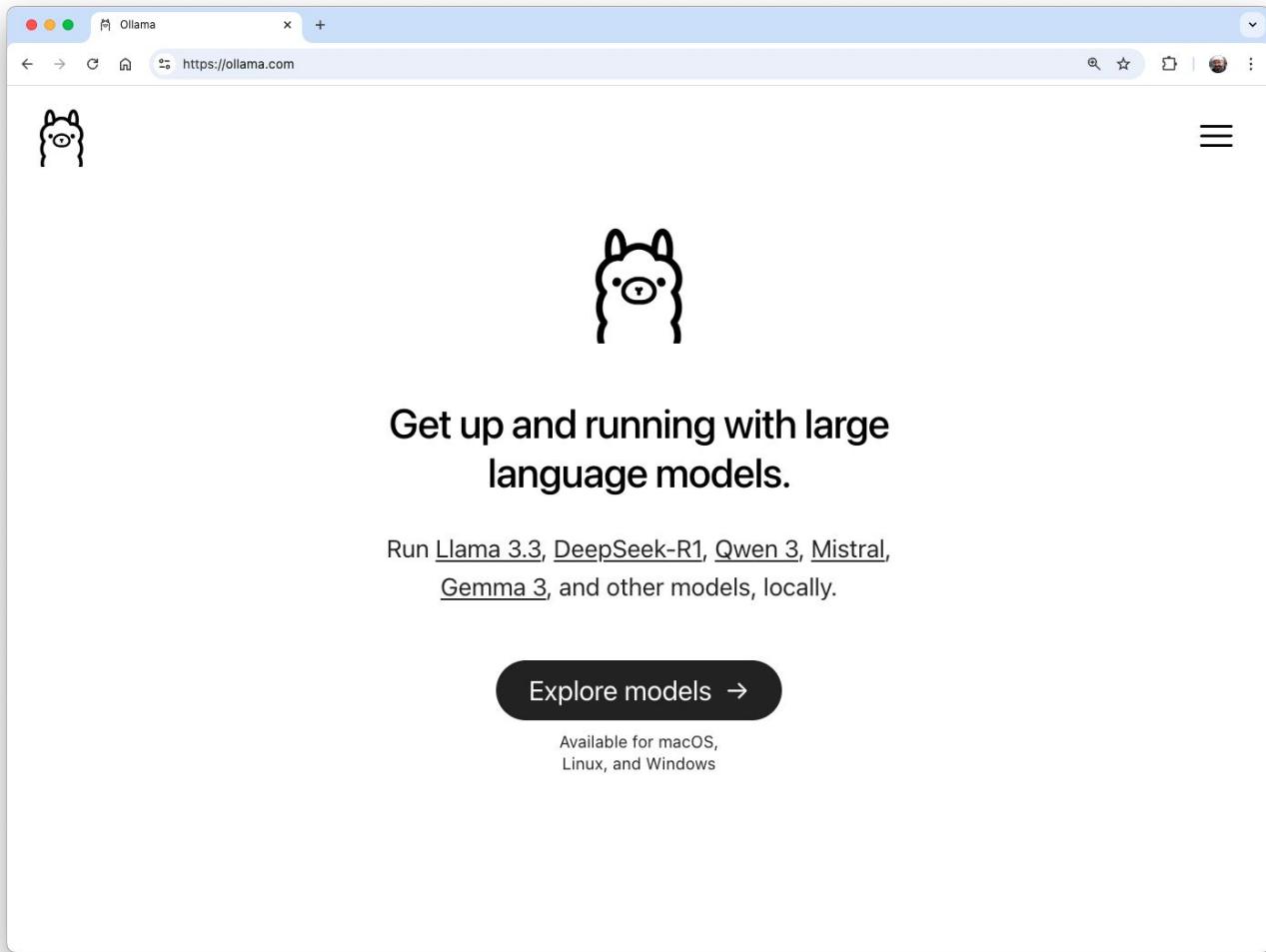
# Transformers to LLMs and SLMs



# Labs – Material



# Running SLMs at the Edge



The primary and most user-friendly tool for running Small Language Models (SLMs) directly on a Raspberry Pi, especially the Pi 5, is Ollama. It is an open-source framework that allows you to install, manage, and run various SLMs (such as TinyLlama, smollm, Microsoft Phi, Google Gemma, Meta Llama, and LLaVa) locally on your Raspberry Pi for tasks like text generation and translation.

## Alternatives options:

[\*\*llama.cpp\*\*](#) and the Hugging Face [\*\*Transformers\*\*](#) library are well-supported for running SLMs on Raspberry Pi. [\*\*llama.cpp\*\*](#) is particularly efficient for running quantized models natively, while Hugging Face [\*\*Transformers\*\*](#) offers a broader range of models and tasks, best suited for smaller architectures due to hardware limitations.

Download Ollama on Linux

https://ollama.com/download/linux



# Download Ollama

macOS      Linux      Windows

For Mac and Windows, download and execute the Ollama file on the machine.

Install with one command:

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

[View script source](#) • [Manual install instructions](#)

```
mjrovai@rpi-5:~  
File Edit Tabs Help  
mjrovai@rpi-5:~ $ python3 -m venv ~/ollama  
mjrovai@rpi-5:~ $ source ~/ollama/bin/activate  
(ollama) mjrovai@rpi-5:~ $ curl https://ollama.ai/install.sh | sh  
% Total    % Received % Xferd  Average Speed   Time     Time      Current  
          Dload  Upload Total   Spent    Left  Speed  
0       0     0      0      0       0      0 --:--:-- --:--:-- --:--:-- 0>>> Inst  
alling ollama to /usr/local  
100 13320    0 13320      0    24659      0 --:--:-- --:--:-- --:--:-- 24621  
>>> Downloading Linux arm64 bundle  
#####
##### 100.0%##0#-#  
#####
##### 100.0%  
>>> Adding ollama user to render group...  
>>> Adding ollama user to video group...  
>>> Adding current user to ollama group...  
>>> Creating ollama systemd service...  
>>> Enabling and starting ollama service...  
>>> The Ollama API is now available at 127.0.0.1:11434.  
>>> Install complete. Run "ollama" from the command line.  
WARNING: No NVIDIA/AMD GPU detected. Ollama will run in CPU-only mode.  
(ollama) mjrovai@rpi-5:~ $
```

Command	Description	Example Usage
<code>ollama serve</code>	Starts the Ollama server for local API access and inference	<code>ollama serve</code>
<code>ollama run &lt;model&gt;</code>	Runs a specified model interactively or with input text	<code>ollama run smolm2:135m</code>
<code>ollama pull &lt;model&gt;</code>	Downloads a model from the Ollama registry	<code>ollama pull phi</code>
<code>ollama list</code>	Lists all models currently downloaded and available	<code>ollama list</code>
<code>ollama show &lt;model&gt;</code>	Displays detailed information about a specific model	<code>ollama show phi</code>
<code>ollama create &lt;new_model&gt; -f &lt;file&gt;</code>	Creates a new model from a Modelfile (customization)	<code>ollama create mymodel -f Modelfile</code>
<code>ollama rm &lt;model&gt;</code>	Removes a specified model from your system	<code>ollama rm phi</code>

Command	Description	Example Usage
<code>ollama ps</code>	Shows currently running Ollama processes or sessions	<code>ollama ps</code>
<code>ollama stop &lt;model&gt;</code>	Stops a running model session or process	<code>ollama stop phi</code>
<code>ollama cp &lt;source&gt; &lt;target&gt;</code>	Copies an existing model to a new name	<code>ollama cp phi mybackup</code>
<code>ollama push &lt;model&gt;</code>	Uploads a model to the Ollama registry (requires account & API key)	<code>ollama push mymodel</code>
<code>ollama --help</code>	Lists all available commands and their descriptions	<code>ollama --help</code>
<code>ollama --version</code>	Shows the current version of Ollama installed	<code>ollama --version</code>
<code>ollama &lt;model&gt; --verbose</code>	Run the model, showing statistics at the end	<code>ollama run phi --verbose</code>

```
mjrovai@raspi-5:~
```

File Edit Tabs Help

```
mjrovai@raspi-5:~ $ ollama list
```

NAME	ID	SIZE	MODIFIED
gemma3:4b	c0494fe00251	3.3 GB	2 months ago
gemma3:1b	2d27a774bc62	815 MB	2 months ago
gemma:2b	b50d6c999e59	1.7 GB	2 months ago
hf.co/tensorblock/TeenyTinyLlama-460m-Chat-GGUF:Q8_0	b3ce44f46784	498 MB	3 months ago
deepseek-r1:1.5b	a42b25d8c10a	1.1 GB	3 months ago
nomic-embed-text:latest	0a109f422b47	274 MB	5 months ago
phi3:latest	4f2222927938	2.2 GB	5 months ago
llama3.2:3b	a80c4f17acd5	2.0 GB	6 months ago
llama3.2:1b	baf6a787fdff	1.3 GB	6 months ago

```
mjrovai@raspi-5:~ $ ollama show llama3.2:3b
```

Model

```
    architecture      llama
    parameters        3.2B
    context length   131072
    embedding length 3072
    quantization     Q4_K_M
```

Capabilities

```
    completion
    tools
```

Parameters

```
    stop      "<|start_header_id|>"
    stop      "<|end_header_id|>"
    stop      "<|eot_id|>"
```

License

```
    LLAMA 3.2 COMMUNITY LICENSE AGREEMENT
    Llama 3.2 Version Release Date: September 25, 2024
```

```
mjrovai@raspi-5:~ $
```

# Small Models

LlaMa



Gemma



```
marcelo_rovai - mjrovai@raspi-5: ~ ssh mjrovai@192.168.4.209 - 74x12
(ollama) mjrovai@raspi-5: ~ $ ollama run llama3.2:3b --verbose
>>> What is the capital of France?
The capital of France is Paris.

total duration: 1.808927736s
load duration: 39.854862ms
prompt eval count: 32 token(s)
prompt eval duration: 221.506ms
prompt eval rate: 144.47 tokens/s
eval count: 8 token(s)
eval duration: 1.506376s
eval rate: 5.31 tokens/s
```

```
marcelo_rovai - mjrovai@raspi-5: ~ ssh mjrovai@192.168.4.209 - 67x13
(ollama) mjrovai@raspi-5: ~ $ ollama run gemma2:2b --verbose
>>> What is the capital of France?
The capital of France is **Paris**. 🎉

total duration: 4.373339337s
load duration: 48.129697ms
prompt eval count: 16 token(s)
prompt eval duration: 1.968114s
prompt eval rate: 8.13 tokens/s
eval count: 13 token(s)
eval duration: 2.313284s
eval rate: 5.62 tokens/s
```

# Ollama Python Library

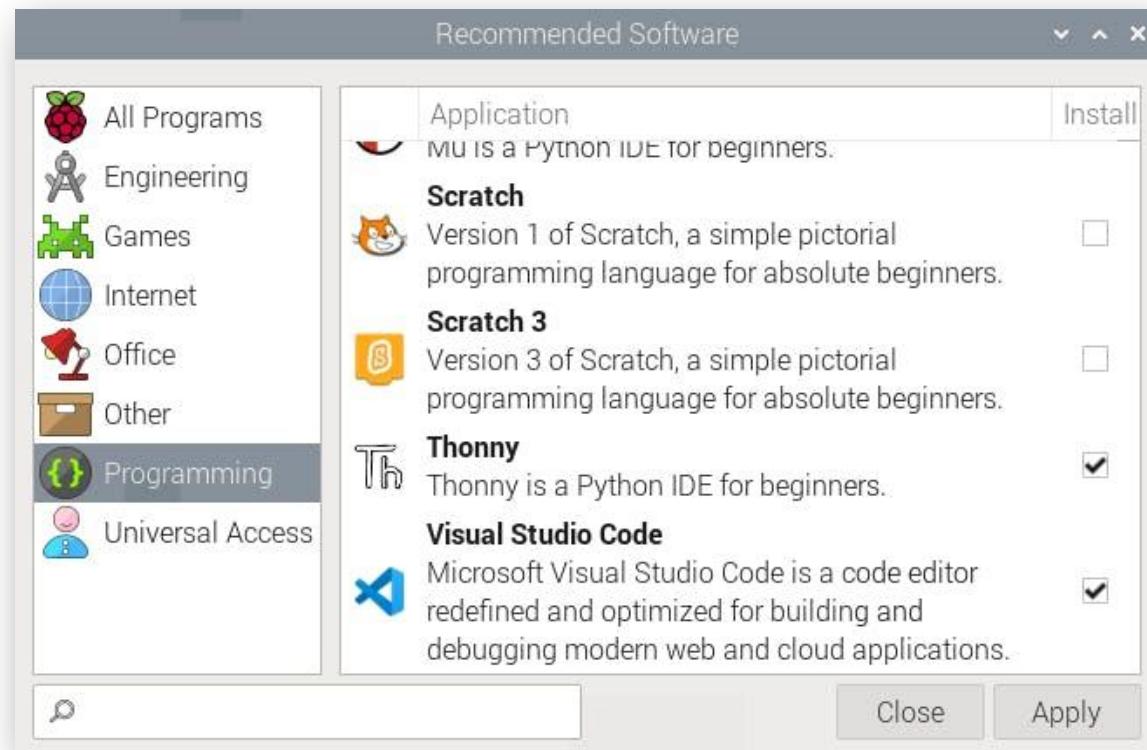
The [Ollama Python library](#) simplifies interaction with advanced LLM models, enabling more sophisticated responses and capabilities, besides providing the easiest way to integrate Python 3.8+ projects with [Ollama](#).

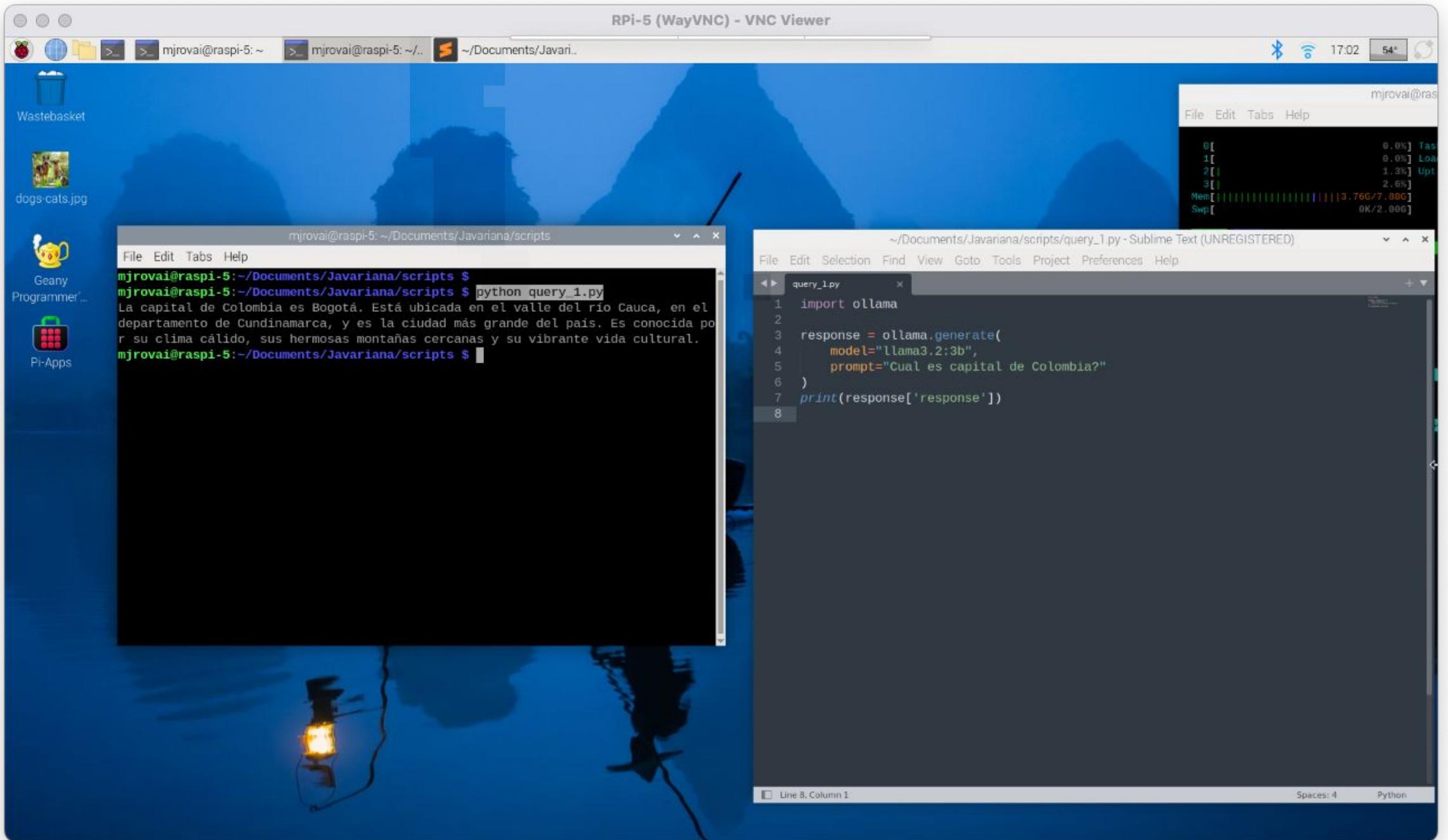
```
$ pip install ollama
```

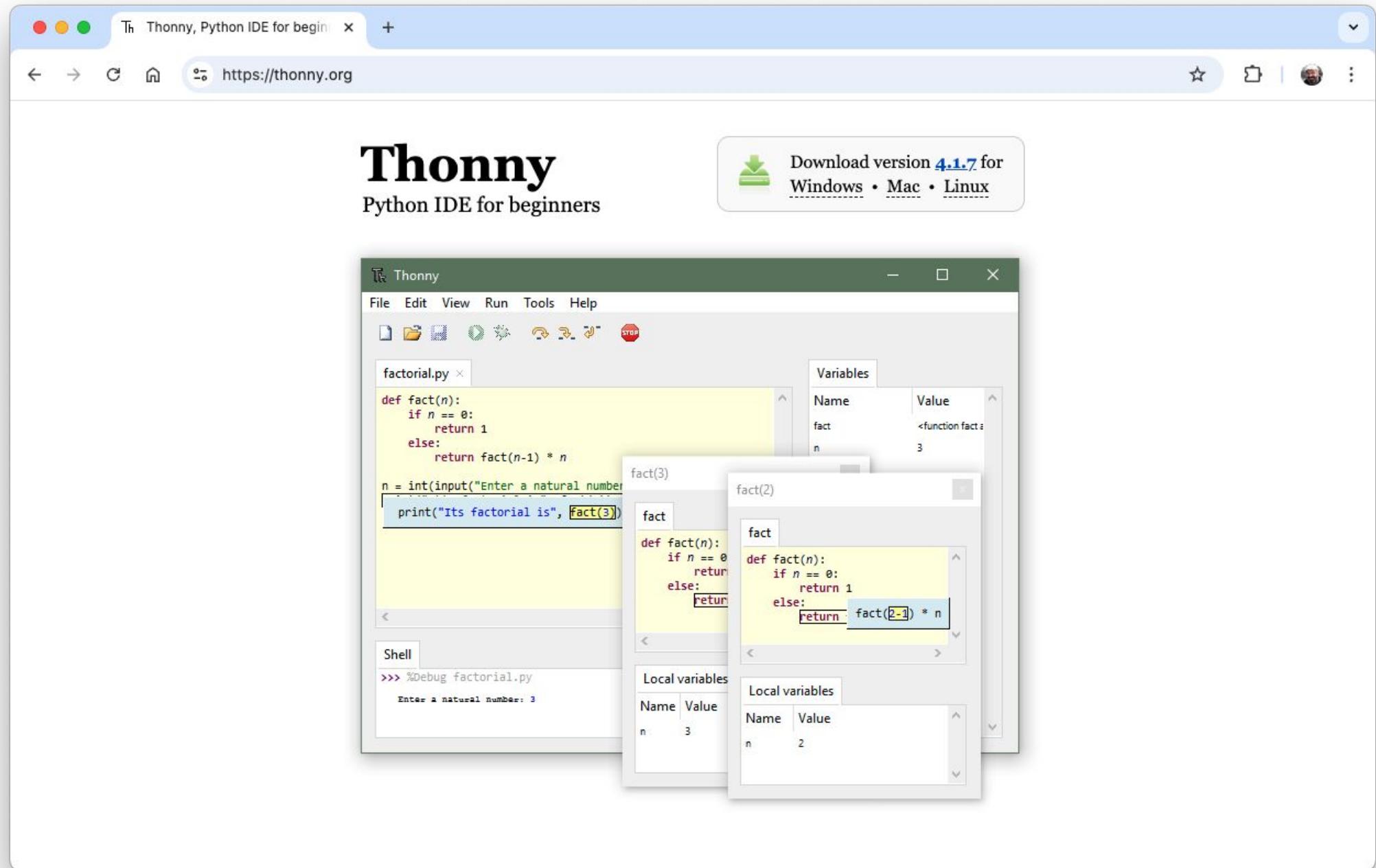
---

## Install/use any IDE

- [Nano](#)
- [Thonny](#)
- [Geany](#)
- [Sublime](#)
- [Jupyter Notebook](#)
- [Visual Studio](#)
- ...







The screenshot shows the Thonny IDE interface. In the top-left code editor, there is a syntax error in the code:

```

import ollama
QUERY = "What is 123456 x 123456?"
```

The code editor has tabs for <untitled> and Shell. In the Shell tab, the user runs:

```

>>> %Run -c $E
Traceback (most recent call last):
  File "<stdin>", line 1
    pip install ollama
      ^
ModuleNotFoundError: No module named 'ollama'
```

An Assistant window is open, displaying the error message: "ModuleNotFoundError: No module named 'ollama' cp\_back.py, line 353".

A Manage packages dialog box is open over the code editor. It shows the search results for "ollama". The "ollama" package is listed with the following details:

- Installed version: 0.4.8
- Installed to: /Users/marcelo\_rovai/Library/Python/3.10/lib/python/site-packages
- Latest stable version: 0.4.8
- Summary: The official Python client for Ollama.
- Author: Ollama
- License: MIT
- PyPI page: <https://pypi.org/project/ollama/>
- Requires: httpx<0.29,>=0.27, pydantic<3.0.0,>=2.9.0

Buttons at the bottom of the dialog include Upgrade, Uninstall, ..., and Close.

## Install Ollama or any Python package on Thonny

Step	Action
1	Open Thonny IDE
2	Tools → Manage Packages
3	Search for <b>ollama</b>
4	Ensure Ollama server is installed & running
5	Pull a model with
6	Use

Thonny - /home/mjrovai/Documents/Javariana/scripts/query\_1.py @ 8:1

The image shows the Thonny Python IDE interface. At the top is a menu bar with icons for New, Load, Save, Run, Debug, Over, Into, Out, Stop, Zoom, Quit, and Support, along with a "Switch to regular mode" link. Below the menu is a tab bar with three tabs: "query\_1.py" (selected), "query\_1\_mem.py", and "query\_2.py". The main area contains the code for "query\_1.py":

```
1 import ollama
2
3 response = ollama.generate(
4     model="llama3.2:3b",
5     prompt="Cual es capital de Colombia?"
6 )
7 print(response['response'])
8
```

Below the code editor is a "Shell" tab containing the output of running the script:

```
>>> %Run query_1.py
La capital de Colombia es Bogotá.
>>> |
```

At the bottom right, it shows "Local Python 3 • /usr/bin/python3" and a menu icon.

# ollama.generate()

The screenshot shows the Thonny Python IDE interface. The top menu bar displays the path "Thonny - /home/mjrovai/Documents/Javariana/scripts/query\_1.py @ 15:28". The toolbar includes icons for New, Load, Save, Run, Debug, Over, Into, Out, Stop, Zoom, Quit, and Support, along with a "Switch to regular mode" button. Two tabs are open: "query\_1.py" and "query\_1\_mem.py". The code in "query\_1.py" is:

```
1 import ollama
2
3 response = ollama.generate(
4     model="llama3.2:3b",
5     prompt="Cual es capital de Colombia?"
6 )
7 print(response['response'])
8
9 response = ollama.generate(
10    model="llama3.2:3b",
11    prompt="Y la de Peru?"
12 )
13
14 print("\nFollow-up response:")
15 print(response['response'])
```

The "Shell" tab at the bottom shows the output of running "query\_1.py":

```
>>> %Run query_1.py
La capital de Colombia es Bogotá.

Follow-up response:
La música peruana es rica y diversa, con una historia que se remonta a la colonia española. A continuación, te presento algunos géneros musicales y artistas destacados de Perú:

Géneros musicales:
1. **Música folclórica**: esta incluye estilos como el marinera, el huayno, el zamacueca y el chalaca, que se originaron en las regiones del sur y central del país.
2. **Rock peruano**: este género surgió en la década de 1980 y se caracteriza por su influencia en la música rock latinoamericana.
3. **Música popular**: esta incluye estilos como el vals, el tango y el balado, que se originaron en las ciudades del Perú.
4. **Música étnica**: esta incluye estilos como el aji triunfo, el huayno y el kuota, que son tradiciones musicales de las comunidades indígenas del país.

Artistas destacados:
1. **Vicente Fernández** (1940-2016): cantante de rock peruano considerado uno de los mayores exponentes de la música peruana.
2. **La India**: cantante y compositora de rock peruano que ha sido una figura importante en la música peruana.
```

The status bar at the bottom indicates "Local Python 3 • /usr/bin/python3".

# ollama.chat()

The screenshot shows the Thonny Python IDE interface. The top menu bar displays the path "Thonny - /home/mjrovai/Documents/Javariana/scripts/query\_1\_mem.py @ 19:36". The toolbar and tabs are identical to the first screenshot. The code in "query\_1\_mem.py" is:

```
1 import ollama
2
3 # Initialize conversation history
4 conversation = []
5
6 # Function to chat with memory
7 def chat_with_memory(prompt):
8     global conversation
9
10    # Add user message to conversation
11    conversation.append({"role": "user", "content": prompt})
12
13    # Generate response with conversation history
14    response = ollama.chat(
15        model="llama3.2:3b",
16        messages=conversation
17    )
18
19    # Add assistant's response to conversation history
20    conversation.append(response["message"])
21
22    # Return just the response text
23    return response["message"]["content"]
24
25 # Example usage
26 prompt = "Cual es la capital de Colombia?"
27 response_text = chat_with_memory(prompt)
28 print(response_text)
29
30 # Ask a follow-up question that relies on memory
31 follow_up = "Y la de Peru?"
32 response_text = chat_with_memory(follow_up)
33 print("\nFollow-up response:")
34 print(response_text)
```

The "Shell" tab at the bottom shows the output of running "query\_1\_mem.py":

```
>>> %Run query_1_mem.py
La capital de Colombia es Bogotá.

Follow-up response:
La capital de Perú es Lima.

>>>
```

The status bar at the bottom indicates "Local Python 3 • /usr/bin/python3".

# LLMs: Optimization Techniques

# Techniques for Enhancing SLM at the Edge

## Fundamentals: Optimizing Prompting Strategies

- Chain-of-Thought Prompting
- Few-Shot Learning
- Task Decomposition

## Intermediate: Building Intelligence Systems

- Building Agents with SLMs
- General Knowledge Router
- Function Calling
- Response Validation

## Advanced: Extending Knowledge and Specialization

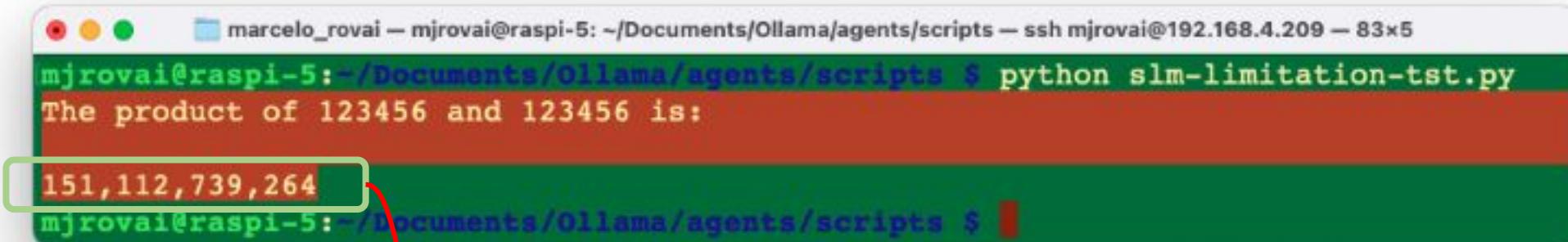
- Retrieval-Augmented Generation (RAG)
- Fine-Tuning for Domain Specialization

**Integration:**  
Combining Techniques for  
Optimal Performance



```
import ollama

response = ollama.generate(
    model="llama3.2:3b",
    prompt="Multiply 123456 by 123456"
)
print(response['response'])
```



A terminal window showing the execution of a Python script named `slm-limitation-tst.py`. The script prompts for the multiplication of 123456 by 123456 and outputs the result 151,112,739,264. The output line is highlighted with a green box.

```
marcelo_rovai — mjrovai@raspi-5: ~/Documents/Ollama/agents/scripts — ssh mjrovai@192.168.4.209 — 83x5
mjrovai@raspi-5:~/Documents/Ollama/agents/scripts $ python slm-limitation-tst.py
The product of 123456 and 123456 is:
151,112,739,264
mjrovai@raspi-5:~/Documents/Ollama/agents/scripts $
```

**WRONG!!!!!!**      The correct answer  $\square 123,456 \times 123,456 = 15,241,383,936$

Thonny - /home/mjrovai/Documents/Javariana/scripts/multiply.py @ 7:28

The screenshot shows the Thonny Python IDE interface. The top menu bar includes 'File' (New, Load, Save), 'Run' (Run, Debug, Over, Into, Out), 'Tools' (Stop, Zoom), 'Help' (Quit, Support), and a language selection dropdown ('Switch to regular mode'). The code editor tab bar shows four files: 'query\_1.py', 'query\_1\_mem.py', 'query\_2.py', and 'multiply.py'. The 'multiply.py' tab is active, displaying the following Python code:

```
1 import ollama
2
3 response = ollama.generate(
4     model="llama3.2:3b",
5     prompt="Cuanto es 123456 multiplicado por 123456?"
6 )
7 print(response['response'])
```

The shell window below shows the execution of the script:

```
>>> %Run multiply.py
¡Hola!
```

Following this, there is a block of Spanish text explaining the mathematical concept of squaring a number:

Para calcular el resultado, podemos utilizar la propiedad de los números enteros que establece que cualquier número entero multiplicado por sí mismo es igual a sí mismo:

$a \times a = a^2$

En este caso, tenemos:

$123456 \times 123456 = 123456^2$

El resultado es:

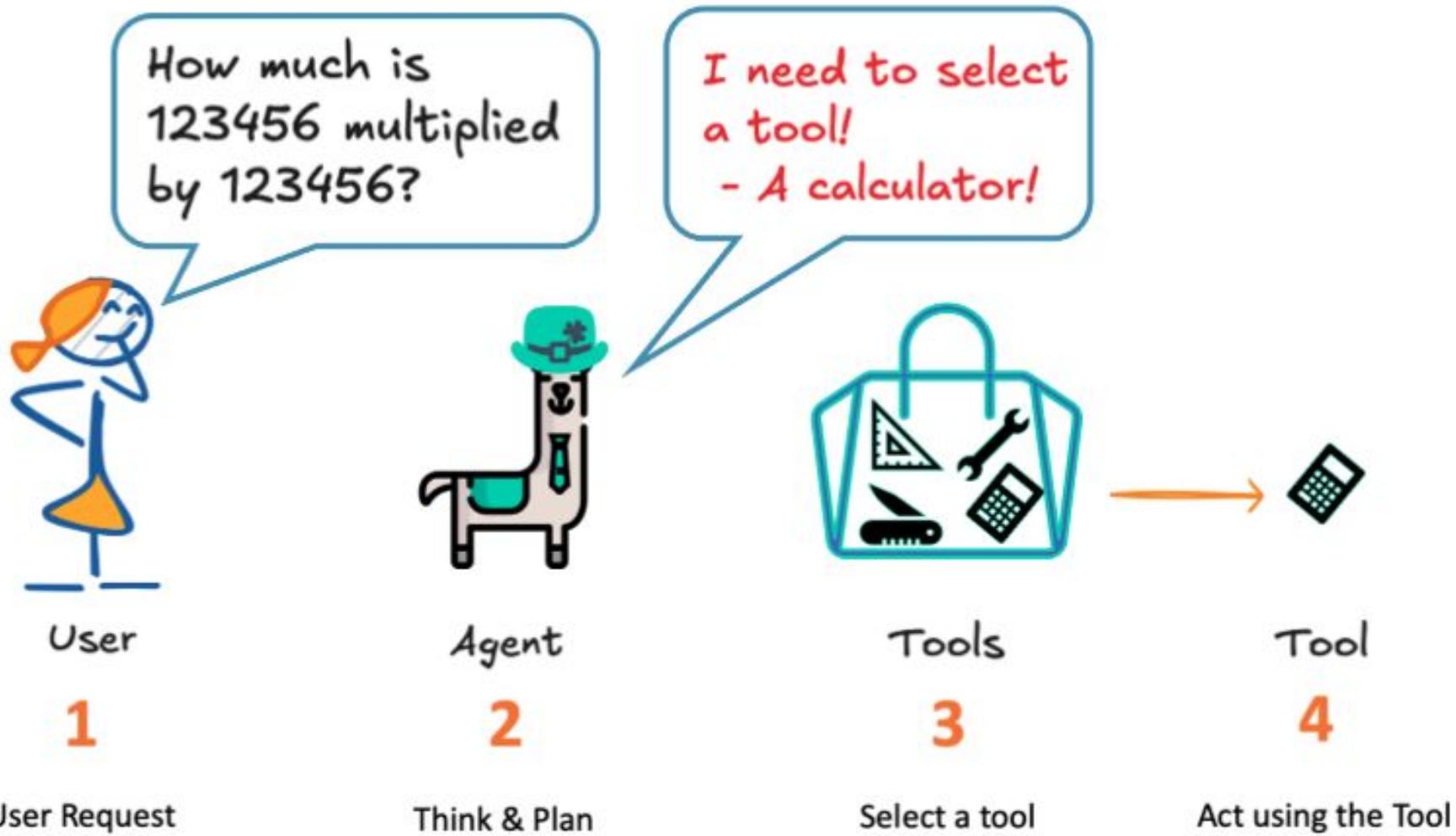
152932457536

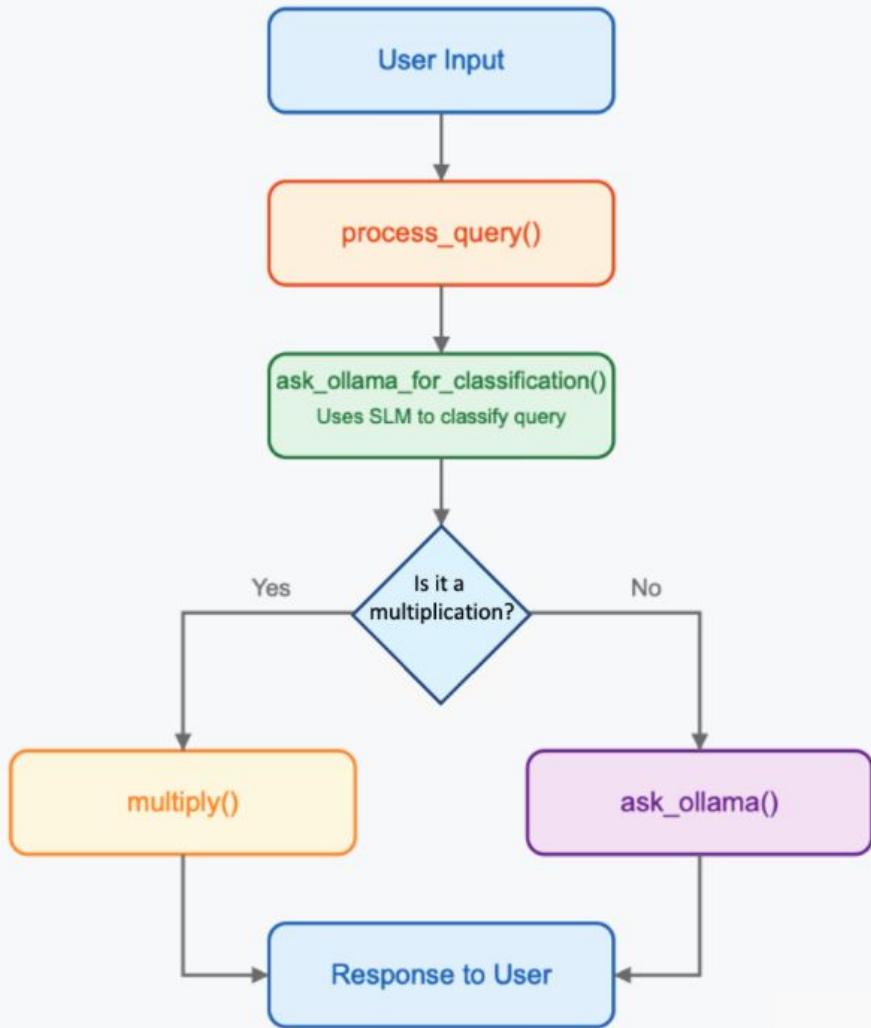
¡Espero que esta respuesta te sea útil!

>>>

At the bottom right, the status bar indicates 'Local Python 3 • /usr/bin/python3'.

# Agents





```

mjrovai@raspi-5:~/Documents/Ollama/agents/scripts$ python 2-simple_agent.py
Ollama Agent (Type 'exit' to quit)
-----
You: Multiply 123456 by 123456
Sending classification request to Ollama
Classification response: {
  "type": "multiplication",
  "numbers": [123456, 123456]
}
Ollama classification: {'type': 'multiplication', 'numbers': [123456, 123456]}
Agent: The product of 123456 and 123456 is 15241383936.
  
```

**It is correct ☐  $123,456 \times 123,456 = 15,241,383,936$**

```

mjrovai@raspi-5:~/Documents/Ollama/agents/scripts$ python 2-simple_agent.py
Ollama Agent (Type 'exit' to quit)
-----
You: What is the capital of Brazil?
Sending classification request to Ollama
Classification response: {
  "type": "general_question"
}
Ollama classification: {'type': 'general_question'}
Sending query to ollama
Agent: The capital of Brazil is Brasília.
You: 
  
```

Thonny - /home/mjrovai/Documents/Javariana/scripts/func\_calling\_3.py @ 152 : 11

New Load Save Run Debug Over Into Out Stop Zoom Quit Support Switch to regular mode

func\_calling.py x func\_calling\_2.py x func\_calling\_3.py x

```
148 # Example usage
149 if __name__ == "__main__":
150     # Set to True to see detailed logging
151     VERBOSE = True
152     main()
```

Shell

```
>>> %Run func_calling_3.py
Ollama Agent (Type 'exit' to quit)
-----
You: Cuanto es 123456 Multiplicado por 123456?
Sending classification request to Ollama
Classification response: {
    "type": "multiplication",
    "numbers": [123456, 123456]
}
Ollama classification: {'type': 'multiplication', 'numbers': [123456, 123456]}

Agent: The product of 123456 and 123456 is 15241383936.

You: Cual es la capital de Colombia?
Sending classification request to Ollama
Classification response: {
    "type": "general_question"
}
Ollama classification: {'type': 'general_question'}

Agent: La capital de Colombia es Bogotá. Está ubicada en el Valle de la Loma, en el departamento de Cundinamarca, al sur del país y es una ciudad importante tanto desde un punto cultural como económico.
```

Local Python 3 • /usr/bin/python3

# Agents Running on SLMs: Overview

Agents powered by Small Language Models (SLMs) are autonomous or semi-autonomous software components that leverage compact, efficient language models to perform specialized tasks, interact with users, or orchestrate workflows. These agents are increasingly favored for their lower computational requirements, faster response times, and suitability for deployment in resource-constrained or privacy-sensitive environments where Large Language Models (LLMs) are impractical.

## Key Traits of SLM-Powered Agents

- Efficiency & Cost-Effectiveness: SLMs require less memory and compute, making agents cheaper to run and easier to deploy at scale or on edge devices.
- Specialization: SLMs can be fine-tuned for specific domains, allowing agents to excel at focused tasks (e.g., compliance, finance, healthcare) without carrying the overhead of generalist LLMs.
- Autonomy & Collaboration: Multiple SLM agents can collaborate, each handling a segment of a workflow, sharing results, and adapting to context for complex, modular automation.
- Adaptability: With techniques like retrieval-augmented generation (RAG) and chain-of-thought prompting, SLM agents can plan, reason, and refine their actions in dynamic environments.

Next page, recommended Packages and Frameworks:

<b>Framework / Package</b>	<b>Description</b>	<b>SLM Support</b>	<b>Key Features</b>
<b>smolagents</b>	Lightweight Python library for building agentic systems with SLMs or LLMs.	Yes	Simple API, tool integration, supports Hugging Face and LiteLLM models, easy function/tool wrapping.
<b>CrewAI</b>	Agent-based framework for multi-agent orchestration.	Yes	Multi-agent workflows, UI tools, monitoring, extensibility, and integrates with many LLM/SLM providers.
<b>Agno</b>	Python framework for converting LLMs/SLMs into agents; supports multiple providers.	Yes	Built-in agent UI, AWS/cloud deployment, database/vector store integration, multi-agent orchestration.
<b>OpenAI Swarm</b>	Open-source multi-agent orchestration framework.	Yes	Lightweight, agent handoff architecture, privacy-focused, built-in retrieval/memory.
<b>Autogen</b>	Open-source framework for multi-agent collaboration and LLM/SLM workflows.	Yes	Cross-language support, local/remote agents, async messaging, scalable, pluggable components.
<b>Arcee Orchestra</b>	Commercial end-to-end agentic AI platform built on SLMs.	Yes	Intelligent model routing, security, compliance, on-prem deployment, fine-tuned SLMs for automation.
<b>LangGraph</b>	Graph-based agent orchestration framework from LangChain for complex, stateful, and multi-agent workflows.	Yes	Cyclic graph workflows, stateful memory (short/long-term), human-in-the-loop, parallelization, subgraphs, reflection/self-correction, visual IDE (LangGraph Studio), persistence, streaming, robust debugging and deployment tools <a href="#">135</a> [7].

# Smolagents (HF)

```
mjrovai@raspi-5:~/Documents/Javariana/scripts $ python agent_calc.py
New run
How is 123456 multiplied by 123456?
LiteLLMModel - ollama/llama3.2:3B Step 1
- Executing parsed code:
result = multiply_calc(a=123456, b=123456)

Out: 15241383936
[Step 1: Duration 279.24 seconds| Input tokens: 2,022 | Output tokens: 66] Step 2
- Executing parsed code:
print("The result of multiplying 123456 by 123456 is:", result)

Execution logs:
The result of multiplying 123456 by 123456 is: 15241383936

Out: None
[Step 2: Duration 33.49 seconds| Input tokens: 4,180 | Output tokens: 119] Step 3
- Executing parsed code:
final_answer(result)

Out - Final answer: 15241383936
[Step 3: Duration 34.30 seconds| Input tokens: 6,479 | Output tokens: 170]
15241383936
mjrovai@raspi-5:~/Documents/Javariana/scripts $
```

```
~/Documents/Javariana/scripts/agent_calc.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
RAG-1-Create-Persistent-Vector-Database.py | RAG-2-Query-the-Persistent-RAG-Database.py | agent_calc.py + ▾
1 from smolagents import CodeAgent, LiteLLMModel, tool
2
3 # Step 1: Define your tool function with a proper docstring
4 @tool
5 def multiply_calc(a: float, b: float) -> float:
6     """Returns the product of two numbers.
7
8     Args:
9         a: The first number to multiply.
10        b: The second number to multiply.
11
12    Returns:
13        float: The product of a and b.
14    """
15    return a * b
16
17 # Step 2: Create the agent
18 agent = CodeAgent(
19     tools=[multiply_calc],
20     model=LiteLLMModel(
21         model_id="ollama/llama3.2:3B",
22         api_base="http://localhost:11434",
23         api_key="ollama",
24         temperature=0.3,
25         num_ctx=4096
26     ),
27     executor_type="local",
28     max_steps=10
29 )
30
31 # Run the agent
32 response = agent.run("How is 123456 multiplied by 123456?")
33 print(response)

Line 17, Column 27
Spaces: 4
Python
```

# Smolagents (Multi-Tools)

```
mjrovai@raspi-5:~/Documents/Javariana/scripts$ python enhanced-agent.py
New run
What is 25 multiplied by 17?
LiteLLMModel - ollama/llama3.2:3B
Step 1
- Executing parsed code:
result = multiply_calc(a=25, b=17)
final_answer(result)

Out - Final answer: 425
[Step 1: Duration 86.74 seconds| Input tokens: 2,073 | Output tokens: 74]
Query: What is 25 multiplied by 17?
Response: 425
```

```
mjrovai@raspi-5:~/Documents/Javariana/scripts
File Edit Tabs Help
New run
What is the capital of Colombia?
LiteLLMModel - ollama/llama3.2:3B
Step 1
- Executing parsed code:
capital = general_knowledge_query(query="Capital of Colombia")
print(capital)

Execution logs:
Bogotá

Out: None
[Step 1: Duration 24.43 seconds| Input tokens: 2,071 | Output tokens: 90]

Step 2
- Executing parsed code:
capital = general_knowledge_query(query="Capital of Colombia")
print(capital)

Execution logs:
Bogotá

Out: None
[Step 2: Duration 60.97 seconds| Input tokens: 4,310 | Output tokens: 225]

Step 3
- Executing parsed code:
capital = general_knowledge_query(query="Capital of Colombia")
print(capital)

Execution logs:
Bogotá

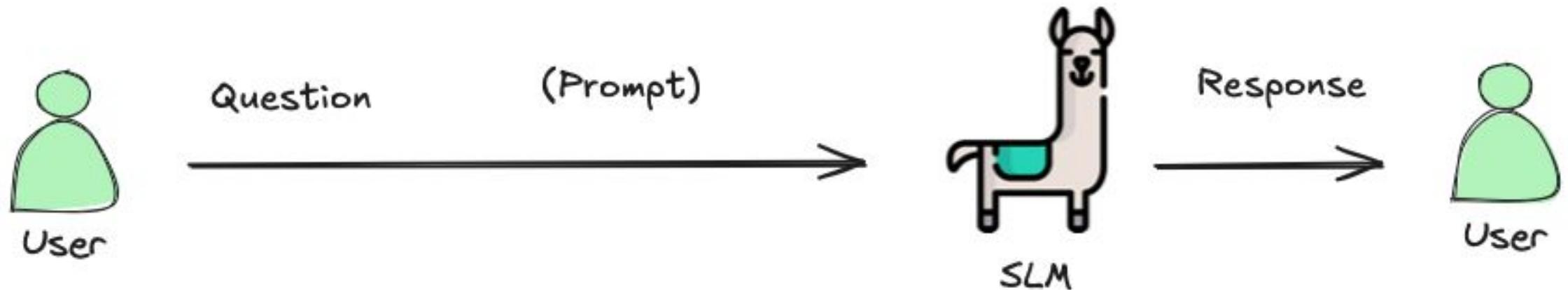
Out: None
[Step 3: Duration 72.32 seconds| Input tokens: 6,762 | Output tokens: 360]

Step 4
- Executing parsed code:
capital = general_knowledge_query(query="Capital of Colombia")
final_answer(capital)

Out - Final answer: Bogotá
[Step 4: Duration 57.28 seconds| Input tokens: 9,427 | Output tokens: 451]
Query: What is the capital of Colombia?
Response: Bogotá
```

# Retrieval-Augmented Generation (RAG)

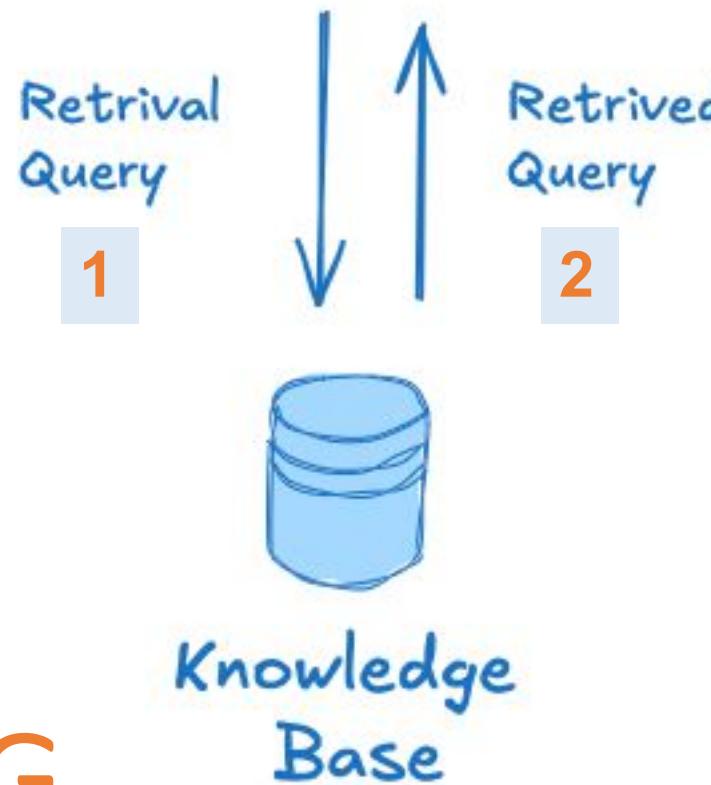
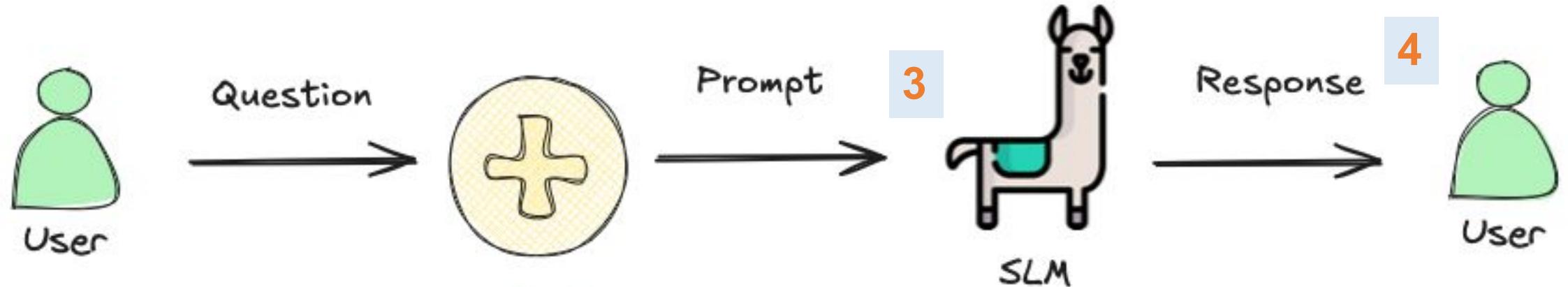
“A method created by the FAIR team at Meta to enhance the accuracy of Large Language Models (LLMs) and reduce false information or “hallucinations.”



```
marcelo_rovai — mjrovai@raspi-5: ~ — ssh mjrovai@192.168.4.209 — 80x13
mjrovai@raspi-5:~$ mjrovai@raspi-5:~$ ollama run llama3 2.3b
>>> What is FOMO
FOMO stands for Fear of Missing Out. It's a common psychological
phenomenon where people feel anxious or apprehensive about potentially
missing out on events, experiences, or social connections that others may
be having.

People with FOMO often experience feelings of insecurity and inadequacy,
worrying that they are not getting the most out of life, nor connecting
with others as much as they should. This can lead to a sense of anxiety,
stress, and even depression.
```

# Usual Prompt



- 1. Query Processing:** When a user asks a question, the system converts it into an embedding (a numerical representation).
- 2. Document Retrieval:** The system searches a knowledge base for documents with similar embeddings.
- 3. Context Enhancement:** Relevant documents are retrieved and combined with the original query.
- 4. Generation:** The SLM generates a response using both the query and the retrieved context.

# With RAG

Thonny - /Users/marcelo\_rovai/Dropbox/2025/90-Workshops-Lectures/Javariana/scripts/RAG-1-Create-Persistent-Vector-Database.py @ 29 : 17

RAG-1-Create-Persistent-Vector-Database.py

```
1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # 1 - Create Persistent Vector Database for RAG
5 # - Edge AI
6
7
8 import warnings
9 import os
10 from langchain.text_splitter import RecursiveCharacterTextSplitter
11 from langchain_community.document_loaders import WebBaseLoader
12 from langchain_community.document_loaders import PyPDFLoader
13 from langchain_llama import LlamaEmbeddings
14 from langchain_community.vectorstores import Chroma
15
16 # Suppress LangSmith warnings
17 warnings.filterwarnings("ignore",
18                         message="API key must be provided when using hosted LangSmith API",
19                         category=UserWarning)

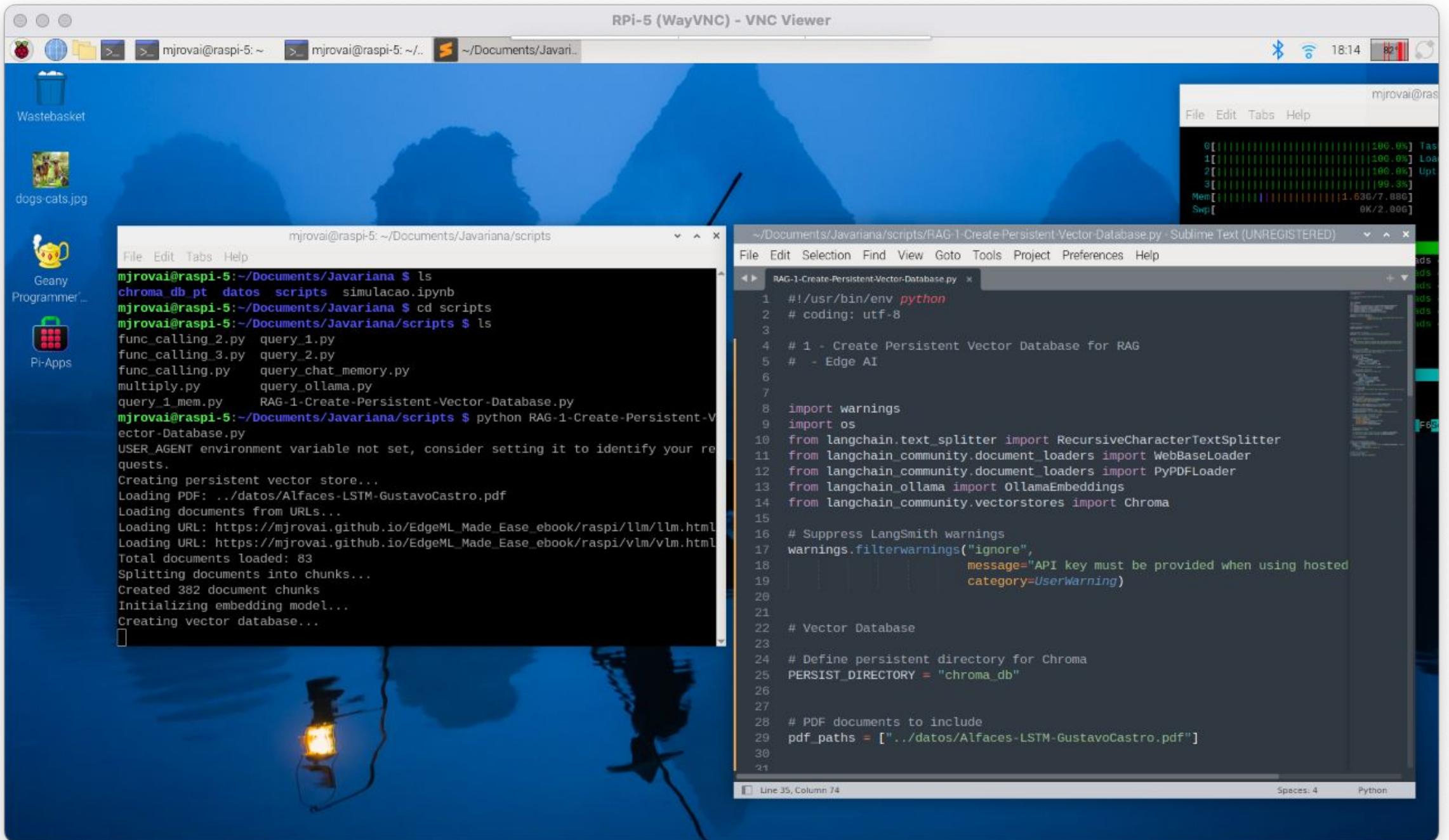
```

Shell

```
>>> %Run RAG-1-Create-Persistent-Vector-Database.py
USER_AGENT environment variable not set, consider setting it to identify your requests.
Creating persistent vector store...
Loading PDF: ./dados/Alfaces-LSTM-GustavoCastro.pdf
Loading documents from URLs...
Loading URL: https://mjrovai.github.io/EdgeML_Made_Easy_ebook/raspi/l1m/l1m.html
Loading URL: https://mjrovai.github.io/EdgeML_Made_Easy_ebook/raspi/vlm/vlm.html
Loading URL: https://mjrovai.github.io/EdgeML_Made_Easy_ebook/raspi/physical_comp/RPi_Physical_Computing.html
Loading URL: https://mjrovai.github.io/EdgeML_Made_Easy_ebook/raspi/iot/sim_iot.html
Total documents loaded: 85
Splitting documents into chunks...
Created 479 document chunks
Initializing embedding model...
Creating vector database...
/Users/marcelo_rovai/Dropbox/2025/90-Workshops-Lectures/Javariana/scripts/RAG-1-Create-Persistent-Vector-Database.py:95: LangChainDeprecationWarning: Since Chroma 0.4.x the manual persist method is no longer supported as docs are automatically persisted.
    vectorstore.persist()
Vector store created and saved to chroma_db
Total document chunks indexed: 479
Database creation complete!
```

Assistant

Analyzing your code ...



File Edit Tabs Help

Your question: Es posible utilizar LSTM?

Generating answer...

Question: Es posible utilizar LSTM?

Retrieving documents...

Retrieved 4 document chunks

Generating answer...

Response latency: 121.66 seconds using model: llama3.2:3b

ANSWER:

=====

Sí, es posible utilizar LSTM en sistemas embebidos debido a su eficiencia energética y capacidad para procesar datos en dispositivos con recursos limitados. La tecnología TinyML ha demostrado la viabilidad de implementar LSTM en plataformas como el ESP32, logrando un balance entre capacidad de predicción y eficiencia energética. Esto se debe a que las técnicas de optimización de hardware y modelos más ligeros permiten implementar LSTM en dispositivos con restricciones de energía y procesamiento críticas.

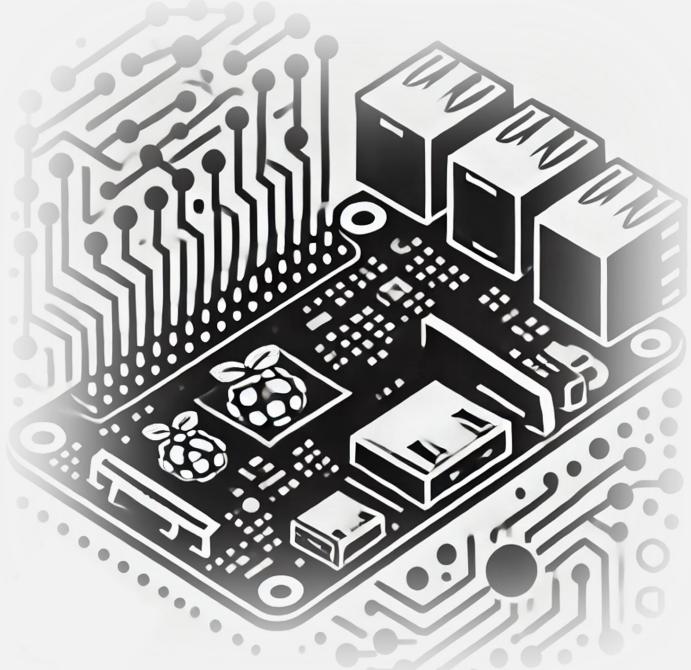
=====

# Homework...

- Physical Computing
- Vision-Language Models

# Edge AI Engineering

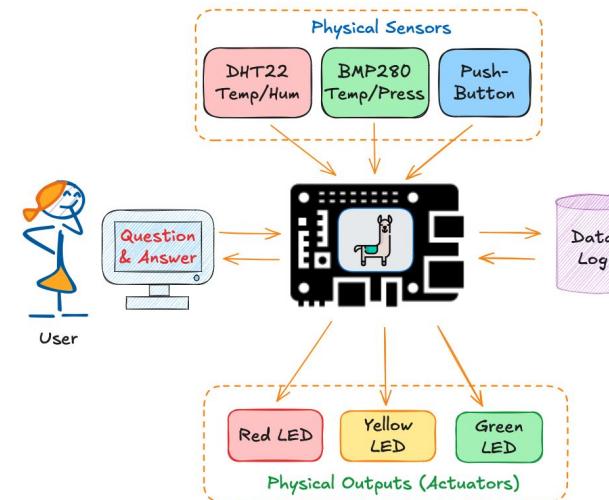
Hands-On with the Raspberry Pi



**Marcelo Rovai**  
Professor Honoris Causa



[EdgeAI Engineering](#)



```
task_prompt = '<CAPTION_TO_PHRASE_GROUNDING>'  
results = run_example(task_prompt, text_input="a church clock tower",image=city)  
plot_bbox(table, results['<CAPTION_TO_PHRASE_GROUNDING>'])
```

[INFO] ==> Florence-2-base (<CAPTION\_TO\_PHRASE\_GROUNDING>), took 12.7 seconds to execute.



# Optional...

- Open WebUI

# Installing and Running the Open Web UI - Raspi

<https://github.com/open-webui/open-webui>

## 1. Install Node.js

```
curl -fsSL https://deb.nodesource.com/setup_20.x | sudo -E bash - &&\nsudo apt-get install -y nodejs
```

## 2. Clone Git:

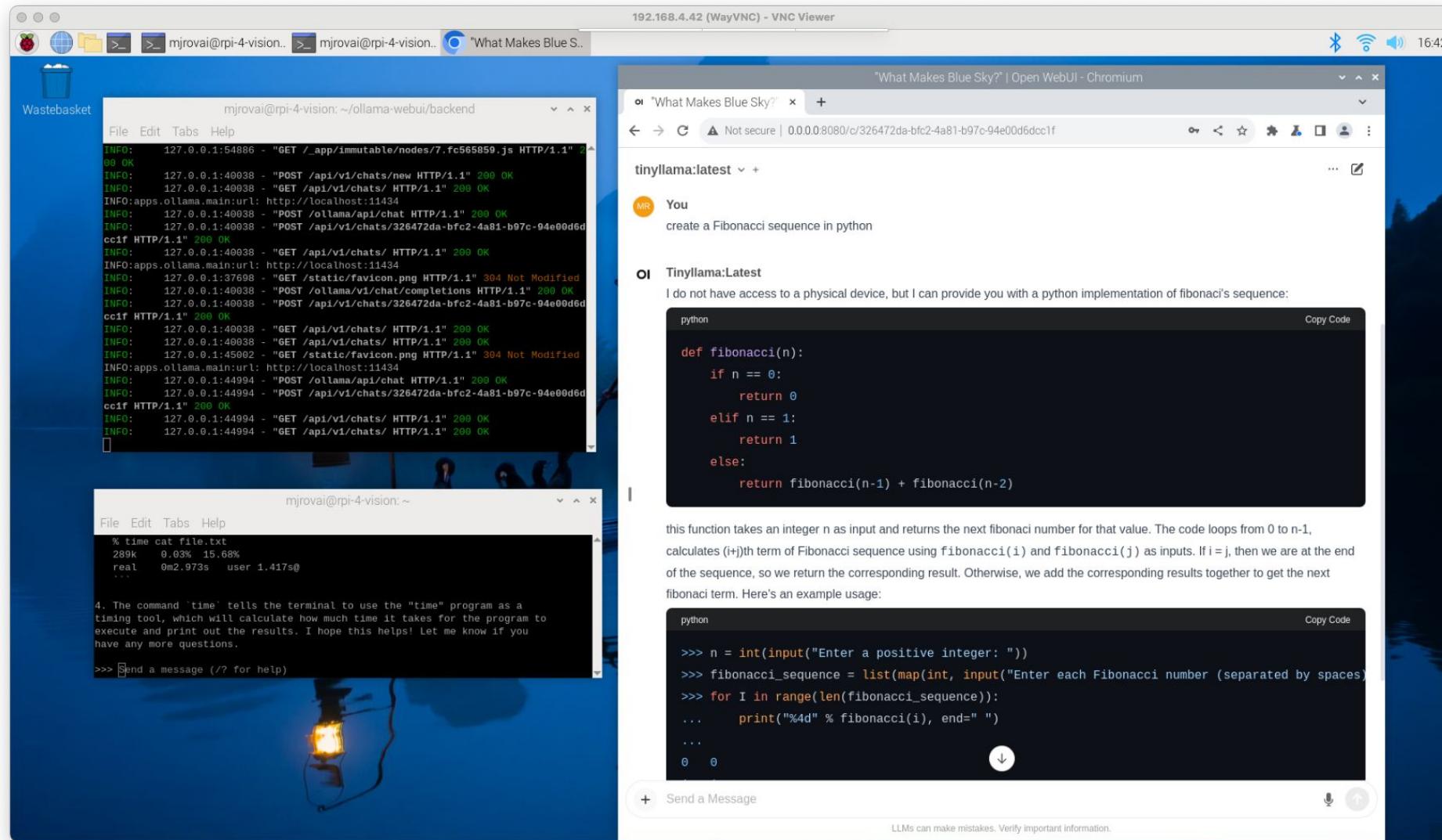
```
git clone https://github.com/ollama-webui/ollama-webui.git\ncd ollama-webui/
```

## 3. Building Frontend Using Node

```
npm i\nnpm run build
```

## 4. Serving Frontend with the Backend

```
cd ./backend\npip install -r requirements.txt --break-system-packages\nsh start.sh
```



Access Ollama Web UI on port 8080 through  
<http://0.0.0.0:8080> on the Raspberry Pi:

# MAC / Windows

1. Install Docker Desktop:

<https://docs.docker.com/get-started/get-docker/>

2. With Ollama installed on your computer, use this command:

```
docker run -d -p 3000:8080 --add-host=host.docker.internal:  
host-gateway -v open-webui:/app/backend/data  
--name open-webui --restart always  
ghcr.io/open-webui/open-webui:main
```

<https://github.com/open-webui/open-webui>

oi ¿Qué pregunta tiene Colombi x +

← → C ⌂ i http://localhost:3000/c/4d569f63-ad85-4034-8068-bc73b65ce95c

☰ oi New Chat ✎ llama3.2:3b + ... ⚙ MR

Workspace

Search

Chats

Untitled

Today

¿Qué pregunta tiene Colombi ...

Capital de Colombia 🇨🇴

Previous 7 days

Cálculo de Multiplicación 📋

Puerto Rico Street Scene

Colombia's Capital City

Previous 30 days

<think> Okay, the user asked how

March

MR Marcelo Rovai + Send a Message

Cual es la capital de Colombia?

La capital de Colombia es Bogotá.

# Questions?

---

Prof. Marcelo J. Rovai

[rovai@unifei.edu.br](mailto:rovai@unifei.edu.br)

UNIFEI - Federal University of Itajuba, Brazil

TinyML4D - Academic Network Co-Chair

EdgeAIP - Academia-Industry Partnership Co-Chair

