ITRI
Industrial Technology
Research Institute

Introduction to LLM (Large Language Model) Applications on AMD Platforms

黃彥勳

©Industrial Technology Research Institute. All rights reserved.

Lesson Outline

Theoretical Component

1. Introduction to Function Calling: Overview
2. Foundations of LLMs
3. Basics of LLM Fine Tuning
4. Retrieval-Augmented Generation (RAG)
5. Function Calling Implementation

Break Time

Practical Component

1. Porting Nvidia Solutions to ROCm/Ryzen AI
2. Preparations for Working With LLMs
3. LLM Deployment Using Transformers
4. LLM Deployment Alternatives
5. Fine Tuning with LoRA
6. Function Calling Using Transformers and Ollama
7. RAG Using Ollama and LM Studio

ITRI
Industrial Technology Research Institute

©Industrial Technology Research Institute. All rights reserved.

Overview

- Introduction

  This presentation serves as an introductory guide for those interested in deploying LLM solutions on ROCm. It covers a high level discussion on the foundation of LLMs and includes practical implementations attendees can follow along.

- Target Audience
  - Programmers familiar with Python
  - Computer Science or Electrical Engineering students interested in building LLM solutions on ROCm
  - Anyone looking to build their first LLM project

- Resources
  - Code and lecture slides: https://github.com/jonathansimple/intro-to-llms

ITRI Industrial Technology Research Institute

©Industrial Technology Research Institute. All rights reserved.

Goals

- Two primary goals:
  1. We want to instruct a LLM to interact with external APIs to perform a task for us (via function calling)
  2. We want to update a LLM's existing knowledge with and without using fine tuning
  – As you will find out, there are many ways to do this
- To accomplish this, we will be discussing a series of LLM related concepts and tools
- To understand why we need to cover these topics, the first step is to understand what function calling is

ITRI Industrial Technology Research Institute

©Industrial Technology Research Institute. All rights reserved.

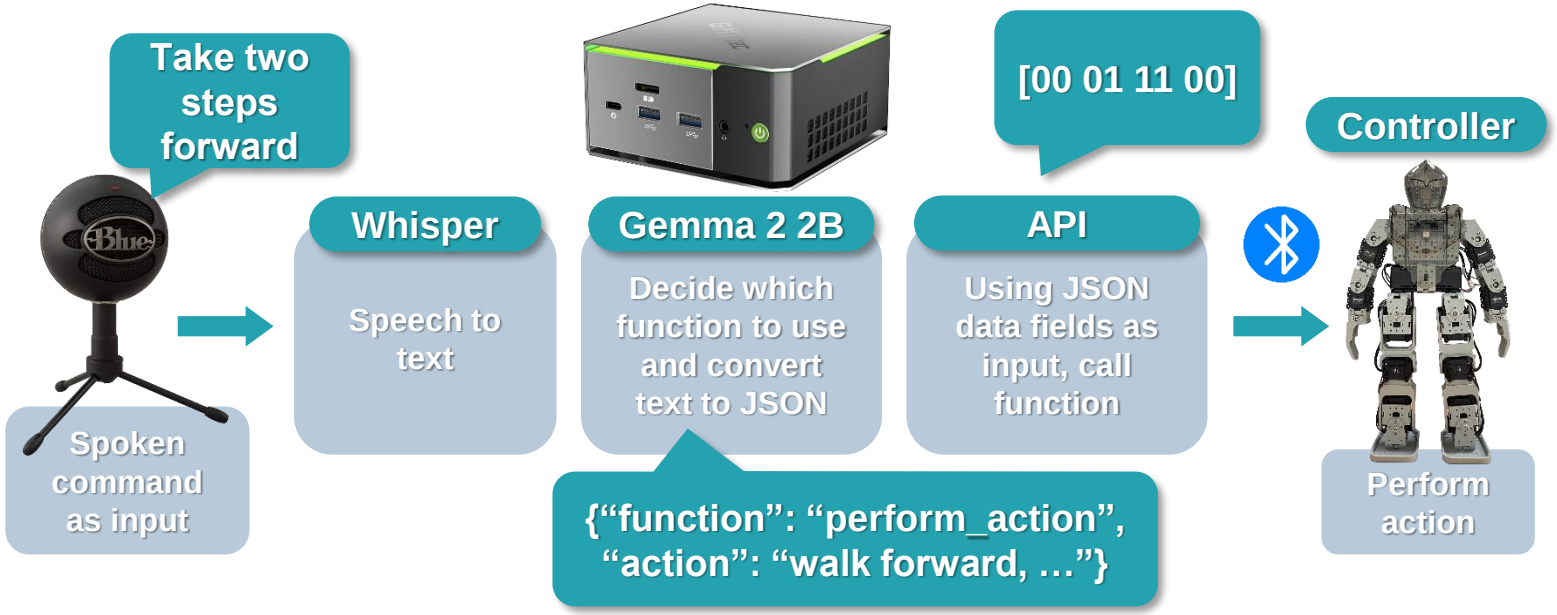
ITRI
Industrial Technology
Research Institute

Introduction to Function Calling: Overview

©Industrial Technology Research Institute. All rights reserved.

Tools/Function Calling

- **Objective:** using the semantic understanding of large language models (LLMs), convert a user's unstructured command into control signals for other applications with function calling
  - For LLMs, function calling refers to when the LLM infers the correct function to call and its input parameters from the user prompt
- **Applications:** IoT, search (retrieve real-time data online or from local database), automated customer service (banking, booking), AI PC



©Industrial Technology Research Institute. All rights reserved.

Essential Components of Function Calling

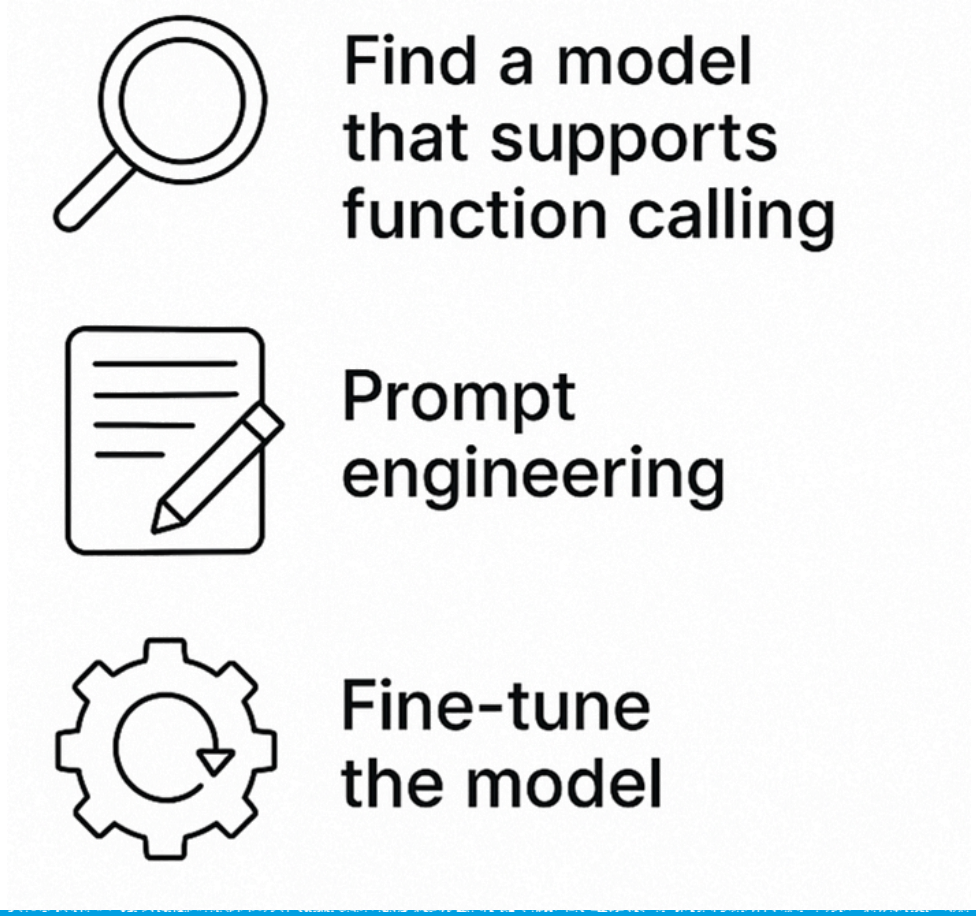
1. **Model:** does the model support function calling?
2. **Prompt:** do the function descriptions describe the task clearly? Is the prompt clear?
3. **External API:** does the application have API support?

ITRI
Industrial Technology Research Institute

©Industrial Technology Research Institute. All rights reserved.

What Happens If You Lack the Essential Components?

Model: the model does not support function calling



GPT, mistral, deepseek, breeze, phi, gemma

(If model has good comprehension) Adjust function descriptions, provide examples for few shot learning

Train the model on public/custom function calling datasets*

©Industrial Technology Research Institute. All rights reserved.

What Happens If You Lack the Essential Components?

Prompt: if the model supports function calling/has good text comprehension, how can you improve function calling performance in the following cases?

1. Function output format is incorrect
   - Few shot learning, use special tokens (<functional>, etc.)
2. Some functions are not getting called/called more than others
   - Check max input size, make sure there isn't too much overlap between functions
3. The model only outputs formatted text and lost the ability to chat normally
   - Adjust system prompt to instruct model to only use function call when applicable, consider using multi-agent systems (one for parsing functions, another for chatting)

©Industrial Technology Research Institute. All rights reserved.

What Happens If You Lack the Essential Components?

External API: does the application have official API support?

✓ Yes

1. Then there's no problem, just pick out the functions you need and determine what parameters to parse as inputs

✗ No

1. Hack the application and write your own API
2. Find alternative applications

ITRI Industrial Technology Research Institute

©Industrial Technology Research Institute. All rights reserved.

ITRI
Industrial Technology
Research Institute

Foundations of LLMs

©Industrial Technology Research Institute. All rights reserved.

Comparison With CNN

- CNN steps: convert image to matrix → inference → f(output)
  - Function f can be argmax for classification, for example

cv2 →
Image augmentation

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

matmul → CNN → $[1 \quad \ldots \quad 0]$ argmax → Label: 0, dog

- LLM steps: tokenize (string to token) → inference (generate) → decode (token to string)
  - **Tokenization example:**

    "ChatGPT is amazing!" → ['Chat', 'G', 'PT', ' is', ' amazing', '!'] → [12345, 678, 91011, 314, 56789, 42]

"How are you?" → tokenize → $[1 \quad \ldots \quad eos]$ → matmul → LLM → $[8 \quad \ldots \quad eos]$ → decode → "I'm doing great"

©Industrial Technology Research Institute. All rights reserved.

Background For LLMs: Tokenizer

- <u>What are tokens?</u>
- LLMs need **tokenizers** to convert strings into tokens (numerical IDs) that the model can understand
  - Working with numerical values (float/int) is easier than strings
  - Every model has their own tokenizer
  - To better generalize unseen vocabulary during training, words are broken down into smaller parts (so one word can be more than one token)

<u>Word seen during training: “unavailable”</u>
“un” → 42
“avail” → 189
“able” → 76

<u>New unknown word seen during inference: “unimaginable”</u>
“un” → 42
“imagine” → ?
“able” → 76

Token 42 = not/opposite
Token 76 = can
So “unimaginable” should have a meaning related to being unable to do something

©Industrial Technology Research Institute. All rights reserved.

What LLMs Are Out There?

- GPT, Deepseek, Llama, gemma, Taide, Breeze, Mistral, Internvl, Qwen…
- Each model has their own strengths and weaknesses:
  - GPT: most well rounded option out there, but not open source and not commercially hostable
  - Breeze: has Traditional Chinese support + open source + commercially viable (3B, 8B); output can be a bit verbose
- More and more recent models are capable of function calling – it's all about picking what suits your use case best (model size, multi-lingual support, etc.)
- Newer models have better text comprehension and some models that weren't trained to do function calling can still perform basic FC if taught in prompts
- Why do we need fine tuning then?
  - Stability, formatting, some smaller models really don't have one-shot learning capability because they can't generalize well enough

ITRI Industrial Technology Research Institute

©Industrial Technology Research Institute. All rights reserved.

ITRI
Industrial Technology
Research Institute

Basics of LLM Fine Tuning

©Industrial Technology Research Institute. All rights reserved.

Fine Tuning Methods

- Most common methods:
  - Full fine tuning
  - LoRA

©Industrial Technology Research Institute. All rights reserved.

Full Fine Tuning

- Like the name suggests, this updates all model parameters during training

Pros:

- Best performance when trying to train a model for a specific task
- Modifies core model behavior (reasoning, output format)
- Used to create foundational models

Cons:

- Very demanding resource wise (time, VRAM, checkpoint storage, data required)
- Easy to overfit on small datasets and forget old skills

©Industrial Technology Research Institute. All rights reserved.

Low-Rank Adaptation (LoRA)

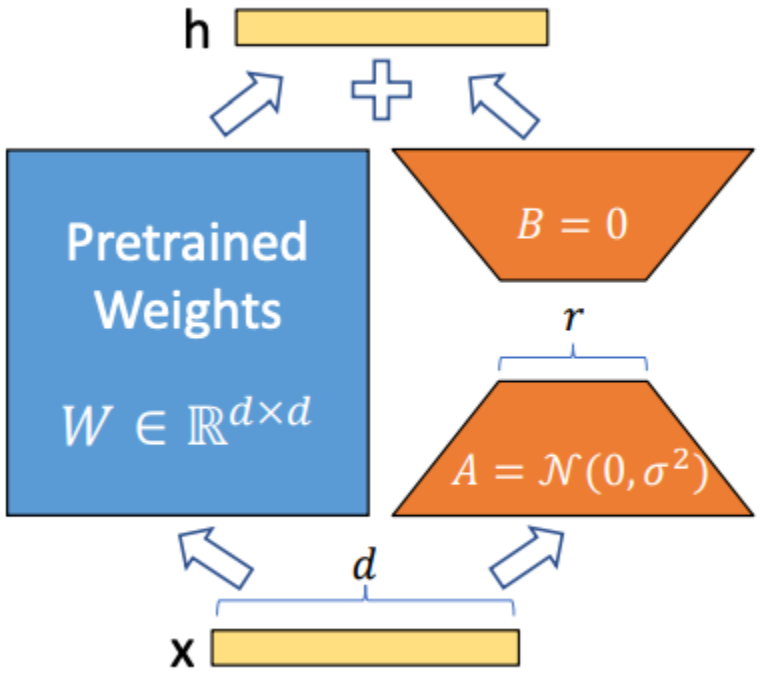
How It Works:

1. Freeze the pre-trained LLM weights.
2. Add small trainable low-rank matrices (A & B) to attention layers.
3. Train only these new matrices, keeping the original model intact.
4. The trainable weights are then merged with the base model.

$$W_{\mathtt{ft}} = W_{\mathtt{pt}} + \underbrace{\Delta W}_{\text{Approximation}} = W_{\mathtt{pt}} + \overbrace{AB}^{\text{Rank Decomposition Matrix}}$$

$$\text{where} \quad W_{\mathtt{ft}}, W_{\mathtt{pt}}, \Delta W, AB \in \mathbb{R}^{d\times d}$$

$$\text{and} \quad \underbrace{A \in \mathbb{R}^{d\times r}, B \in \mathbb{R}^{r\times d}}_{\text{Low Rank}}$$



©Industrial Technology Research Institute. All rights reserved.

Low-Rank Adaptation (LoRA)

Key Benefits:

- ☑ **Reduces Memory Usage** – Fine-tunes **only a small fraction** of the model parameters.
- ☑ **Faster Training** – Requires fewer computations, making it cost-efficient.
- ☑ **Retains Pre-Trained Knowledge** – Keeps most of the original model weights frozen.
- ☑ **Easier Adaptation** – Fine-tune **large models** (e.g., LLaMA, Mistral) on **consumer GPUs**.
- ☑ **No Added Inference Latency** – doesn't slow down model.

Use Cases:

- Fine-tuning LLMs on **domain-specific data** (e.g., medical, legal, finance) and **special formatting**
- Fine tuning on limited hardware
- When you only want to partially modify model performance while maintaining core behavior

ITRI Industrial Technology Research Institute

©Industrial Technology Research Institute. All rights reserved.

ITRI
Industrial Technology
Research Institute

Retrieval-Augmented Generation (RAG)

©Industrial Technology Research Institute. All rights reserved.

What Is RAG (Retrieval-Augmented Generation)?

Problem: LLMs like GPT are trained on massive datasets, but they:

- Can hallucinate (make up facts)
- Do not have any knowledge of current events after their training cutoff date
- Can't access private or custom data during pretraining
- For smaller models, they lack domain specific expertise

Solution: RAG

- Step 1: Retrieve relevant documents from an external database (e.g., PDFs, websites)
- Step 2: Generate an answer using both the user question and the retrieved content

This makes responses:

- ☑ More accurate
- ☑ More up-to-date
- ☑ Customizable to your domain

All without fine-tuning!

ITRI Industrial Technology Research Institute

©Industrial Technology Research Institute. All rights reserved.

RAG Encoder

- How does the retrieval system know which documents are **relevant**?

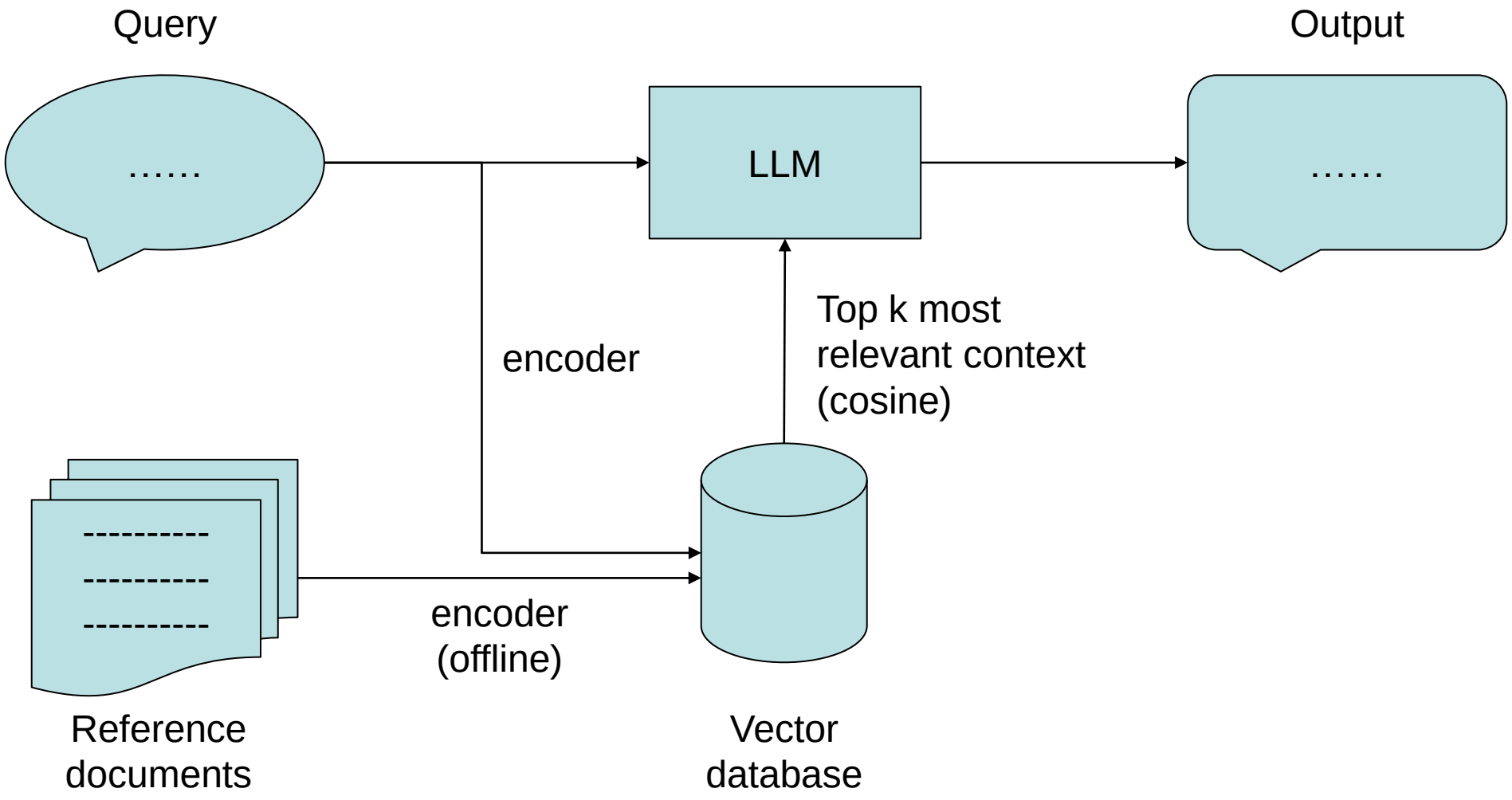
- With RAG, you are working (generally) with 2 models
  - LLM model (generator) that answers your questions
  - Embedding model (encoder) that maps your question into the same vector space as your dataset (vector store)

- The vector store finds the most similar documents (top-k) to the query vector (cosine similarity)

ITRI Industrial Technology Research Institute

©Industrial Technology Research Institute. All rights reserved.

An Example RAG Framework



©Industrial Technology Research Institute. All rights reserved.

RAG At Work

If you use an older model like gemma-2-2b-it and ask it:

Do you have any information on who the president of the United States is in 2025?

It will respond with:

I do not have access to real-time information, including future events. Therefore, I cannot tell you who the President of the United States will be in 2025.

With RAG:

*According to the news sources you provided, the President of the United States in 2025 would be **Donald Trump**.*

©Industrial Technology Research Institute. All rights reserved.

When to Use RAG?

- Most tasks can be accomplished with RAG (without fine tuning the model)
  - To draw a parallel with CNNs, RAG is similar to using data augmentation (prompt engineering) and few shot learning to circumvent fine tuning
- Applications:
  - Domain specific QA: customer service, automatic inspection
  - Personalized service: can't be expected to load entire chat history or fine tune model for every user
  - Cheaply update model while maintaining model's core behavior

ITRI Industrial Technology Research Institute

©Industrial Technology Research Institute. All rights reserved.

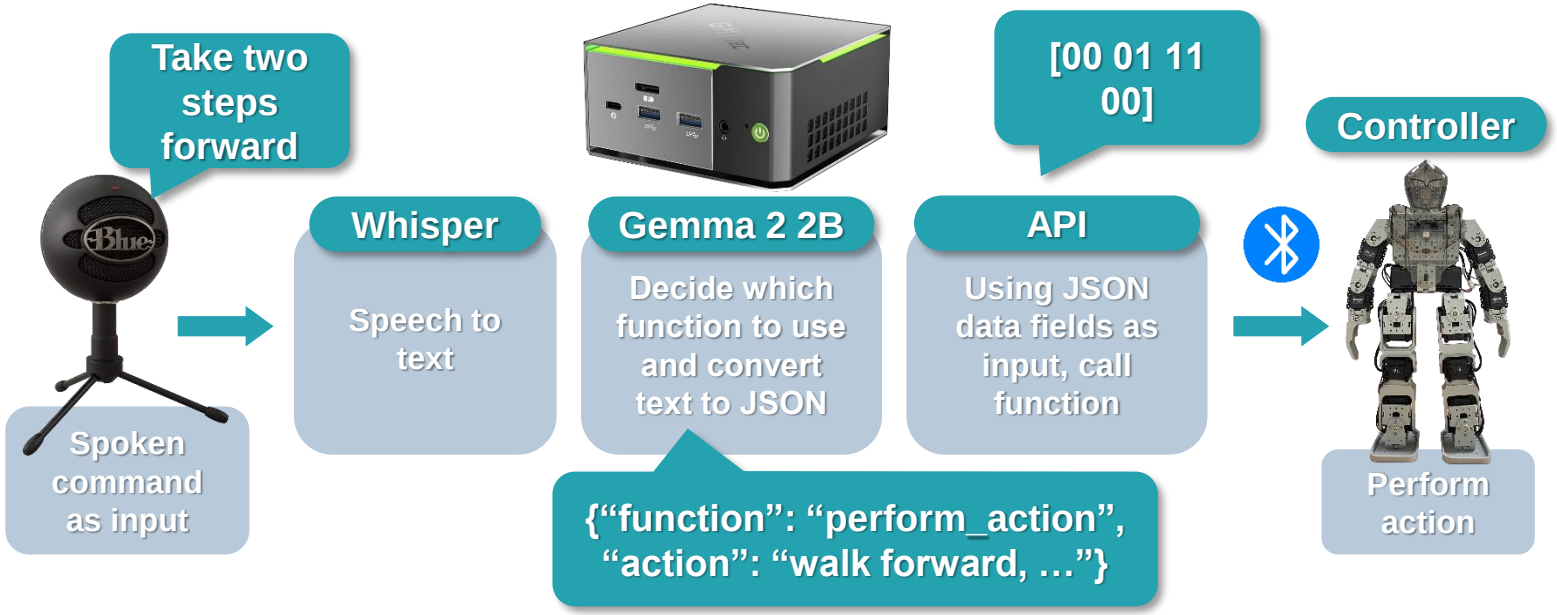
ITRI
Industrial Technology
Research Institute

Function/Tool Calling: Using Natural Language to Interface with External APIs

©Industrial Technology Research Institute. All rights reserved.

Tools/Function Calling

- **Objective:** using the semantic understanding of large language models (LLMs), convert a user's unstructured command into control signals for other applications with function calling
  - For LLMs, function calling refers to when the LLM infers the correct function to call and its input parameters from the user prompt
- **Applications:** IoT, search (retrieve real-time data online or from local database), automated customer service (banking, booking), AI PC



©Industrial Technology Research Institute. All rights reserved.

Our Model Can Call Functions, Now What?

☑ **Model:** does the model support function calling?

❓ **Prompt:** do the function descriptions describe the task clearly? Is the prompt clear?

❓ **External API:** does the application have API support?

➢ Before we craft our prompt, we need to know what task we are performing.

ITRI Industrial Technology Research Institute

©Industrial Technology Research Institute. All rights reserved.

Our Model Can Call Functions, Now What?

- Decide on which API you are using:
  - Example: Open-Meteo (free open source weather forecast API)
- Pick out the functions you will use:
  - Example: get_weather(latitude, longitude)
  - Input: (12, 12)
  - Output: json with **current time** and **temperature**
- API supports other functions like wind speed, hourly temperature, etc.

Open-Meteo's API returns an easily parsed json

- **Next step:** crafting functions/tools for LLM to use (description + prompts = use these APIs)

ITRI Industrial Technology Research Institute

©Industrial Technology Research Institute. All rights reserved.

Function Calling Prompt Crafting

A prompt for function calling contains the following:

- General direction/role assignment
- Function list
  - Function name
  - Description
  - Function parameter names
  - Function parameter descriptions
  - Required inputs
- (Optional) Desired output format

©Industrial Technology Research Institute. All rights reserved.

Function Calling Prompt Crafting

- General direction/role assignment
  - Example: “You are a helpful assistant with access to the following functions. Use them if required - ”
- Assigning a role to a LLM can influence its output:
  - This affects tonal style, how strict/loose the model should be when deciding whether it should use functions or chat normally
  - Example: “You are a reliable, trustworthy expert on semiconductor manufacturing. You can answer anything related to your domain of expertise but you must decline to answer when asked about other topics. When applicable, use the functions provided below to help answer the query:”

ITRI Industrial Technology Research Institute

©Industrial Technology Research Institute. All rights reserved.

Function Calling Prompt Crafting

- Function list: a function list includes all of the functions you want the LLM to be able to call
  - You can think of this as a header file for the LLM
  - Comments are **IMPORTANT**

©Industrial Technology Research Institute. All rights reserved.

Function Calling Prompt Crafting

- A single function in the list will have these parameters:
  - **Function name:** name of the function
  - **Description:** what the function does, when it should be used, how it should be used; as a rule of thumb, if an intern can't tell how this function works, the LLM won't be able to either
  - **Function parameter names:** the names of the inputs to the function
  - **Function parameter descriptions:** function input properties (bool, enum), can include textual description for complex use cases
  - **Required inputs:** which inputs must be included
- Function output is not explicitly written in the parameters (but usually included in function description)
  - This is because the LLM's job is to turn natural language into f(x), the API is what handles the output generation

ITRI Industrial Technology Research Institute

©Industrial Technology Research Institute. All rights reserved.

Function Calling Prompt Crafting

- Function declaration example:
  - **Function name:** get_weather
  - **Description:** Get the current temperature for the provided coordinates in Celsius.
  - **Function parameter names:** latitude, longitude
  - **Function parameter descriptions:** {"type": "number"} (additional comments and enumerator rule not needed)
  - **Required inputs:** latitude, longitude

©Industrial Technology Research Institute. All rights reserved.

Function Calling Prompt Crafting

- Desired output format:
  - Some LLMs are already trained on function calling datasets and can output structured data
  - Sometimes, it can still be useful to specify a particular output format to improve stability
- Recommended ways to format outputs:
  - Json
  - Token delimiters
- Example:

  If a function should be used, respond with the function name and its parameters in json format like so:

  {"name": "function_name", "arguments": {"arg_1": "value_1", "arg_2": "value_2", ...}}

ITRI Industrial Technology Research Institute

©Industrial Technology Research Institute. All rights reserved.

Putting Everything Together

You are a helpful assistant with access to the following functions. Use them if required -

```
{
    "name": "get_weather",
    "description": "Get the current temperature for the provided coordinates in Celsius.",
    "parameters": {
        "type": "object",
        "properties": {
            "latitude": {"type": "number"},
            "longitude": {"type": "number"}
        },
        "required": ["latitude", "longitude"]
    }
}
```

If a function should be used, respond with the function name and its parameters in json format like so:

{"name": "function_name", "arguments": {"arg_1": "value_1", "arg_2": "value_2", ...}}

User Query:

©Industrial Technology Research Institute. All rights reserved.

A More Complex Example

You are a helpful assistant with access to the following functions. Use them if required -

```
{
    "name": "get_action_set",
    "description": "Get the series of actions and the number of times each action needs to be performed in order",
    "parameters": {
        "type": "object",
        "properties": {
            "actions": {
                "type": "array",
                "description": "The actions that need to be performed in the order that is specified by the user. If an action has to be performed multiple times, then repeat the action entry for the same number specified by the user, e.g. "Take two steps forward" would turn into ["walk forward", "walk forward"]." ,
                "items": {
                    "type": "string",
                    "enum": ["bow", "cheer", "scratch head", "handstand", "sit", "stand", "stabilize", "walk forward", "walk backward", "walk left", "walk right", "turn right", "turn left"],
                }
            },
        },
        "required": ["actions"]
    }
}
```

Descriptions are useful when the parameter has special rules or is unintuitive (generally good to have regardless)

Enums are useful when you only expect a fixed set of values in the field

If a function should be used, respond with the function name and its parameters in json format like so:

{"name": "function_name", "arguments": {"arg_1": "value_1", "arg_2": "value_2", ...}}

User Query:

©Industrial Technology Research Institute. All rights reserved.

We Have Our Model and Prompt, Now What?

☑ **Model:** does the model support function calling?

☑ **Prompt:** do the function descriptions describe the task clearly? Is the prompt clear?

❓ **External API:** does the application have API support?

➢ The easy part. Integrate the LLM pipeline with your chosen API

ITRI Industrial Technology Research Institute

©Industrial Technology Research Institute. All rights reserved.

System Integration

- Prepare your API calls to take in the inputs you specified

```python
import requests

def get_weather(latitude, longitude):
    response = requests.get(f"https://api.open-meteo.com/v1/forecast?latitude={latitude}&longitude={longitude}&current=temperature_2m")
    data = response.json()
    return data['current']['temperature_2m']
```

← Prepare the API functions you require

```python
start_time = time.time()
function_calling_output = model.generate(
        input_ids = inputs,
        **generation_config
    )
end_time = time.time()

generated_tokens = function_calling_output.shape[-1] - inputs.shape[-1]
elapsed_time = end_time - start_time
tokens_per_sec = generated_tokens / elapsed_time

function_calling_output = tokenizer.decode(function_calling_output[0],skip_special_tokens=True)

print("==> Gemma output:", function_calling_output)

temperature = get_weather(function_calling_output)
print(f"The current temperature is {temperature} Celsius")
print(f"Generated {generated_tokens} tokens in {elapsed_time:.2f}s → {tokens_per_sec:.2f} tokens/sec")
```

← LLM output

← API call using LLM output as parameters

Don’t worry about the other arcane code, we’ll cover everything here in the practical section

©Industrial Technology Research Institute. All rights reserved.

Break Time

- Q&A
- Session resumes in 10 minutes
- Practical portion next:
  - Code walkthrough + demos
  - Feel free to clone the repo for code we'll be going through. If you are setting up a fresh environment, it may take more time than the break would allow. If you already have a working environment with GPU acceleration (CUDA or ROCm), you can follow along by running the code on your own machine
- Github link: https://github.com/jonathansimple/intro-to-llms

ITRI Industrial Technology Research Institute

©Industrial Technology Research Institute. All rights reserved.

ITRI
Industrial Technology
Research Institute

Porting Nvidia Solutions to ROCm/Ryzen AI

©Industrial Technology Research Institute. All rights reserved.

AMD Hardware and Driver Support

- Two types of AMD hardware:
  - AI PCs (laptops) - Ryzen AI 300-series PCs with iGPU + NPU
  - Desktops – GPUs like RX 6800 and W6800 and up

- What's the difference?
  - Different hardware require different drivers:
  - AI PCs: NPU drivers + Ryzen AI Software
  - Desktops: ROCm (Linux), HIP SDK* (Windows)

*HIP (Heterogeneous-computing Interface for Portability) SDK for Windows brings a subset of the ROCm platform to Windows

ITRI Industrial Technology Research Institute

©Industrial Technology Research Institute. All rights reserved.

Determining Which ROCm Version Is For You

Short answer: depends on your hardware

1. For the full compatibility matrix, visit: https://rocm.docs.amd.com/en/latest/compatibility/compatibility-matrix.html

2. For which AMD GPUs supports which LLVM target, visit: https://rocm.docs.amd.com/projects/install-on-linux/en/latest/reference/system-requirements.html

2.

AMD Instinct | AMD Radeon PRO | AMD Radeon

GPU	Architecture	LLVM target	Support
AMD Radeon RX 7900 XTX	RDNA3	gfx1100	✅
AMD Radeon RX 7900 XT	RDNA3	gfx1100	✅
AMD Radeon RX 7900 GRE	RDNA3	gfx1100	✅
AMD Radeon VII	GCN5.1	gfx906	⚠️

Compatibility matrix

2025-05-07 33 min read time

Applies to Linux

1.

Use this matrix to view the ROCm compatibility and system requirements across successive major and minor releases.

You can also refer to the past versions of ROCm compatibility matrix.

Accelerators and GPUs listed in the following table support compute workloads (no display information or graphics). If you're using ROCm with AMD Radeon or Radeon Pro GPUs for graphics workloads, see the Use ROCm on Radeon GPU documentation to verify compatibility and system requirements.

ROCm Version	6.4.0	6.3.3	6.2.0
Operating systems & kernels	Ubuntu 24.04.2	Ubuntu 24.04.2	Ubuntu 24.04
	Ubuntu 22.04.5	Ubuntu 22.04.5	Ubuntu 22.04.5, 22.04.4
	RHEL 9.5, 9.4	RHEL 9.5, 9.4	RHEL 9.4, 9.3
	RHEL 8.10	RHEL 8.10	RHEL 8.10, 8.9
	SLES 15 SP6	SLES 15 SP6, SP5	SLES 15 SP6, SP5
	Oracle Linux 9, 8 [1]	Oracle Linux 8.10 [1]	Oracle Linux 8.9 [1]
	Debian 12 [2]	Debian 12 [2]	
GPU / LLVM target	gfx1100	gfx1100	gfx1100
	gfx1030	gfx1030	gfx1030
	gfx942	gfx942	gfx942 [3]
	gfx90a	gfx90a	gfx90a
	gfx908	gfx908	gfx908

ITRI Industrial Technology Research Institute

©Industrial Technology Research Institute. All rights reserved.

ROCm Installation (Ubuntu 24.04)

Install ROCm

```
wget https://repo.radeon.com/amdgpu-install/6.4/ubuntu/noble/amdgpu-install_6.4.60400-1_all.deb
sudo apt install ./amdgpu-install_6.4.60400-1_all.deb
sudo apt update
sudo apt install python3-setuptools python3-wheel
sudo usermod -a -G render,video $LOGNAME # Add the current user to the render and video groups
sudo apt install rocm
```

Install AMD GPU drivers

```
wget https://repo.radeon.com/amdgpu-install/6.4/ubuntu/noble/amdgpu-install_6.4.60400-1_all.deb
sudo apt install ./amdgpu-install_6.4.60400-1_all.deb
sudo apt update
sudo apt install "linux-headers-$(uname -r)" "linux-modules-extra-$(uname -r)"
sudo apt install amdgpu-dkms
```

Information from:
https://rocm.docs.amd.com/projects/install-on-linux/en/latest/install/quick-start.html

©Industrial Technology Research Institute. All rights reserved.

ROCm Installation for Different Distributions

ROCm installation

Ubuntu | Debian | Red Hat Enterprise Linux | Oracle Linux | SUSE Linux Enterprise Server | Azure Linux

24.04 | 22.04

```
wget https://repo.radeon.com/amdgpu-install/6.4/ubuntu/noble/amdgpu-install_6.4.60400-1_all.del
sudo apt install ./amdgpu-install_6.4.60400-1_all.deb
sudo apt update
sudo apt install python3-setuptools python3-wheel
sudo usermod -a -G render,video $LOGNAME # Add the current user to the render and video groups
sudo apt install rocm
```

Information from:
https://rocm.docs.amd.com/projects/install-on-linux/en/latest/install/quick-start.html

©Industrial Technology Research Institute. All rights reserved.

ROCm Pytorch Setup (Ubuntu 22.04)

. Pull the selected image.

```
docker pull rocm/dev-ubuntu-22.04:latest
```

- Python 3.10

Start a Docker container using the downloaded image.

```
docker run -it --device=/dev/kfd --device=/dev/dri --group-add video rocm/dev-ubuntu-22.04:l
```

Install the required dependencies for the wheels package.

```
sudo apt update
sudo apt install libjpeg-dev python3-dev python3-pip
pip3 install wheel setuptools
```

Install `torch`, `torchvision`, and `torchaudio`, as specified in the installation matrix.

Note

The following command uses the ROCm 6.2 PyTorch wheel. If you want a different version of ROCm, modify the command accordingly.

```
pip3 install --pre torch torchvision torchaudio --index-url https://download.pytorch.org/whl/nigl
```

Screenshots from: https://rocm.docs.amd.com/projects/install-on-linux/en/latest/install/3rd-party/pytorch-install.html

--index-url https://download.pytorch.org/whl/nightly/rocm6.2/

ITRI Industrial Technology Research Institute

©Industrial Technology Research Institute. All rights reserved.

AI PC NPU Driver Installation (Windows)

- Go to: https://ryzenai.docs.amd.com/en/latest/inst.html#install-npu-drivers

Prerequisites

The Ryzen AI Software supports AMD processors with a Neural Processing Unit (NPU). Consult the release notes for the full list of supported configurations.

The following dependencies must be present on the system before installing the Ryzen AI Software:

Dependencies	Version Requirement
Windows 11	build >= 22621.3527
Visual Studio	2022
cmake	version >= 3.26
Anaconda or Miniconda	Latest version

Ensure your system has the prerequisites installed

©Industrial Technology Research Institute. All rights reserved.

AI PC NPU Driver Installation (Windows)

Install NPU Drivers

- Download the NPU driver installation package `NPU Driver`
- Install the NPU drivers by following these steps:
  - Extract the downloaded `NPU_RAI1.4_GA_257_WHQL.zip` zip file.
  - Open a terminal in administrator mode and execute the `.\npu_sw_installer.exe` exe file.
- Ensure that NPU MCDM driver (Version:32.0.203.257, Date:3/12/2025) is correctly installed by opening `Device Manager` -> `Neural processors` -> `NPU Compute Accelerator Device`.

Install Ryzen AI Software

- Download the RyzenAI Software installer `ryzen-ai-1.4.0.exe`.
- Launch the MSI installer and follow the instructions on the installation wizard:
  - Accept the terms of the Licence agreement
  - Provide the destination folder for Ryzen AI installation (default: `C:\Program Files\RyzenAI\1.4.0`)
  - Specify the name for the conda environment (default: `ryzen-ai-1.4.0`)

The Ryzen AI Software packages are now installed in the conda environment created by the installer.

Create an AMD account and download + run the installers (free)

ITRI Industrial Technology Research Institute

©Industrial Technology Research Institute. All rights reserved.

AI PC NPU Driver Installation (Linux)

Other Installation Options

Linux Installer

Compiling BF16 models requires more processing power than compiling INT8 models. If a larger BF16 model cannot be compiled on a Windows machine due to hardware limitations (e.g., insufficient RAM), an alternative Linux-based compilation flow is supported. More details can be found here: Ryzen AI Software on Linux

A Linux installer can be found at the bottom of
https://ryzenai.docs.amd.com/en/latest/inst.html

©Industrial Technology Research Institute. All rights reserved.

Porting Nvidia-Based Solutions to ROCm

- **No code change needed**
  - The only difference between a CUDA and a ROCm project is in its environment setup
- Specifying devices:
  - CPU: “cpu”
  - GPU: “cuda”
  - Device is specified using the same method: “0”, “0,1”, etc.

ITRI Industrial Technology Research Institute

©Industrial Technology Research Institute. All rights reserved.

Porting Nvidia-Based Solutions to AI PCs

- **Minor code change needed**
  - AI PCs rely on the onnx runtime to enable NPU acceleration, without it, speedup is limited
  - Projects with CPU-only fallback solutions do not need to be modified
  - When setting up this environment, I **strongly recommend** following AMD's Ryzen AI Software + NPU driver installation (the Ryzen AI Software installer will set up a compatible conda environment for you)

ITRI Industrial Technology Research Institute

©Industrial Technology Research Institute. All rights reserved.

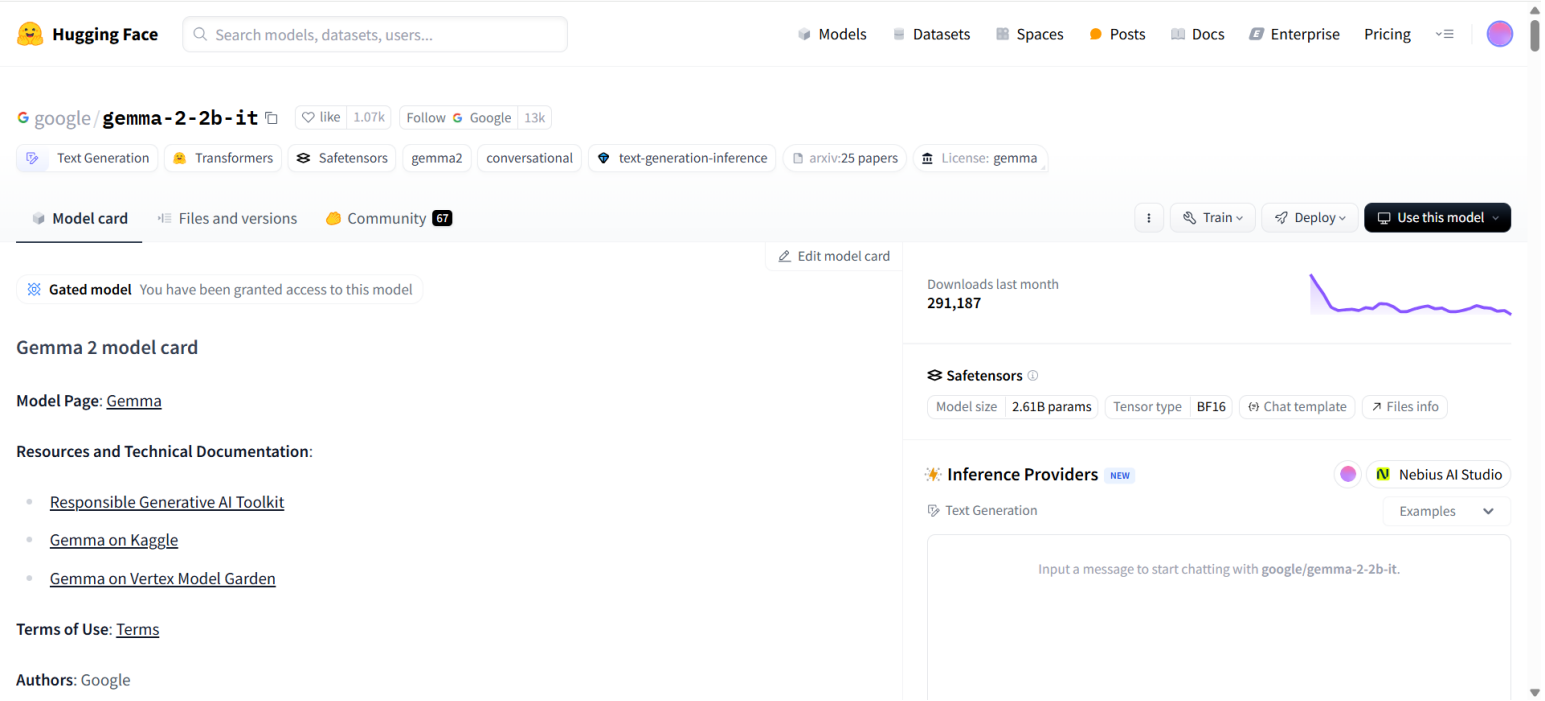
ITRI
Industrial Technology
Research Institute

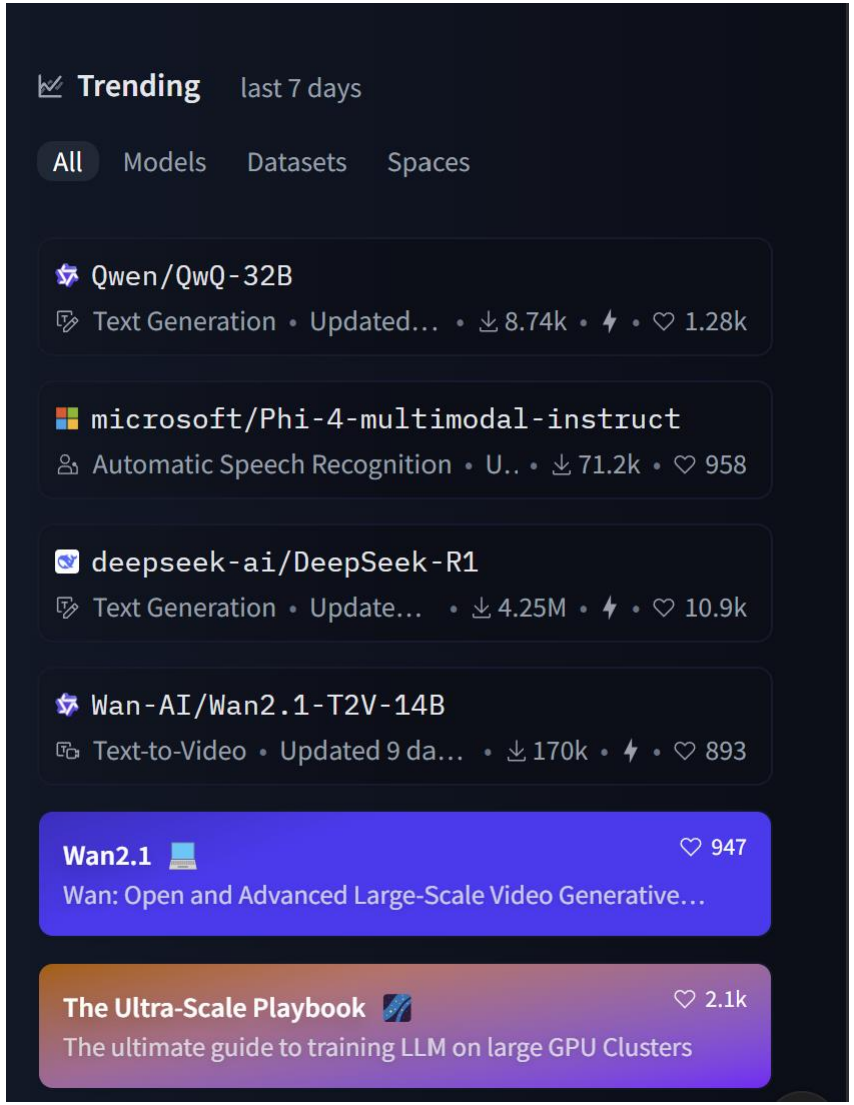
Practical Component: Preparations for Working With LLMs

©Industrial Technology Research Institute. All rights reserved.

Downloading LLMs

- Where to download models: https://huggingface.co/
- You can find trending (new) models on the home page

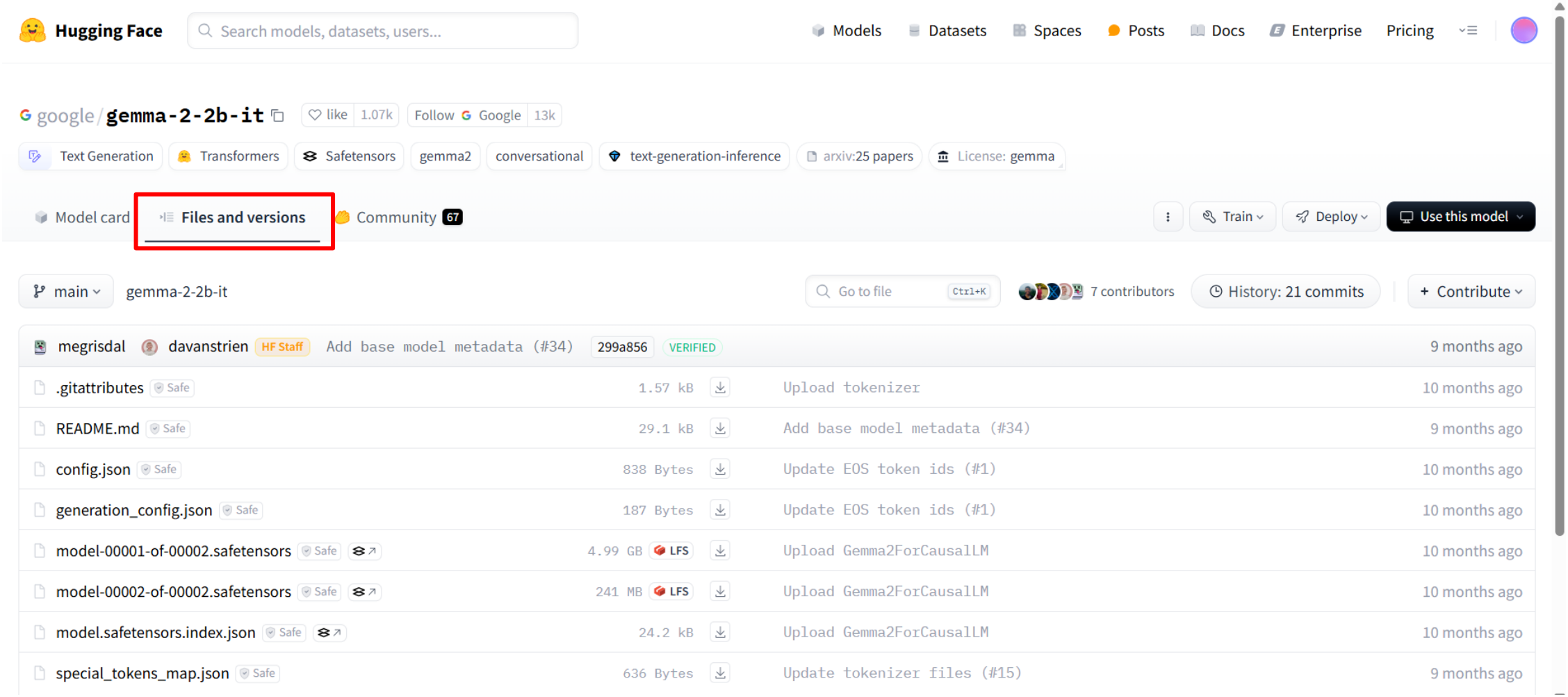




Example of what a model's Huggingface page looks like

©Industrial Technology Research Institute. All rights reserved.

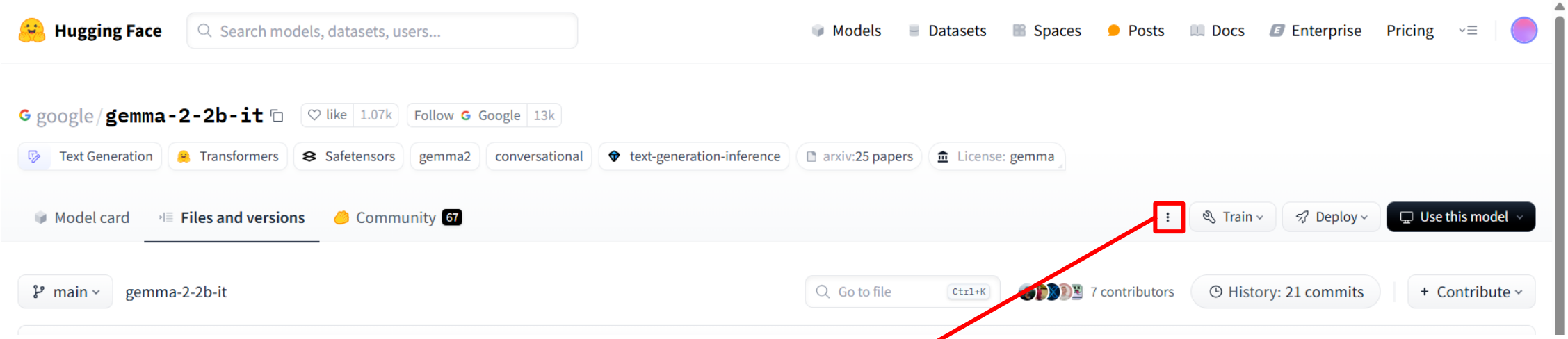
Downloading LLMs (1) – Manual Download

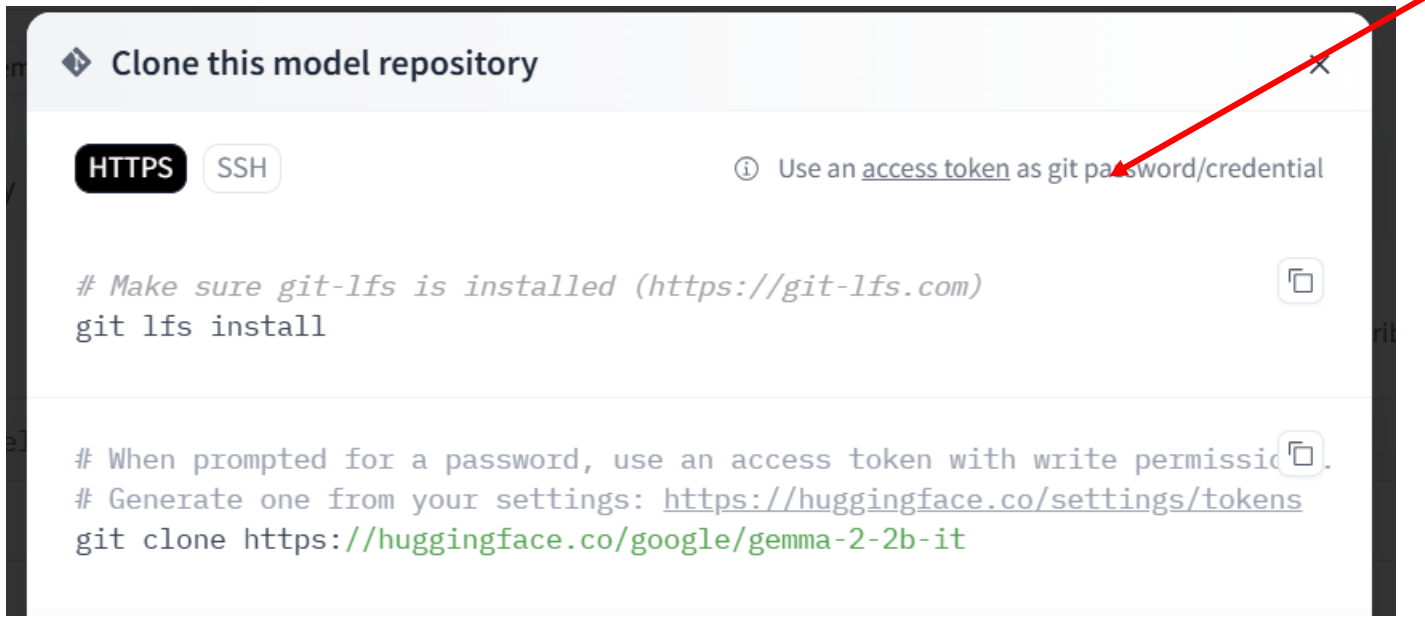


ITRI Industrial Technology Research Institute

©Industrial Technology Research Institute. All rights reserved.

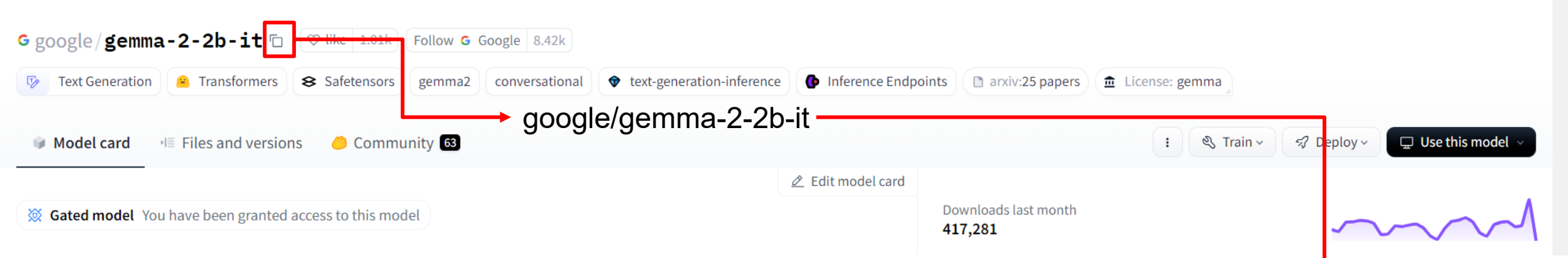
Downloading LLMs (2) - Git





©Industrial Technology Research Institute. All rights reserved.

Downloading LLMs (3) – Huggingface



Running models hosted on Huggingface (this will save the models inside the Huggingface library folder)**:**

tokenizer = AutoTokenizer.from_pretrained(**"google/gemma-2-2b-it"**, trust_remote_code=True)

model = AutoModelForCausalLM.from_pretrained(**"google/gemma-2-2b-it"**, trust_remote_code=True, device_map=0)

Running local/downloaded model:

tokenizer = AutoTokenizer.from_pretrained(**"/home/sean/Desktop/qlora-main/gemma2-2b-it"**, trust_remote_code=True)

model = AutoModelForCausalLM.from_pretrained(**"/home/sean/Desktop/qlora-main/gemma2-2b-it"**, trust_remote_code=True, device_map=0)

©Industrial Technology Research Institute. All rights reserved.

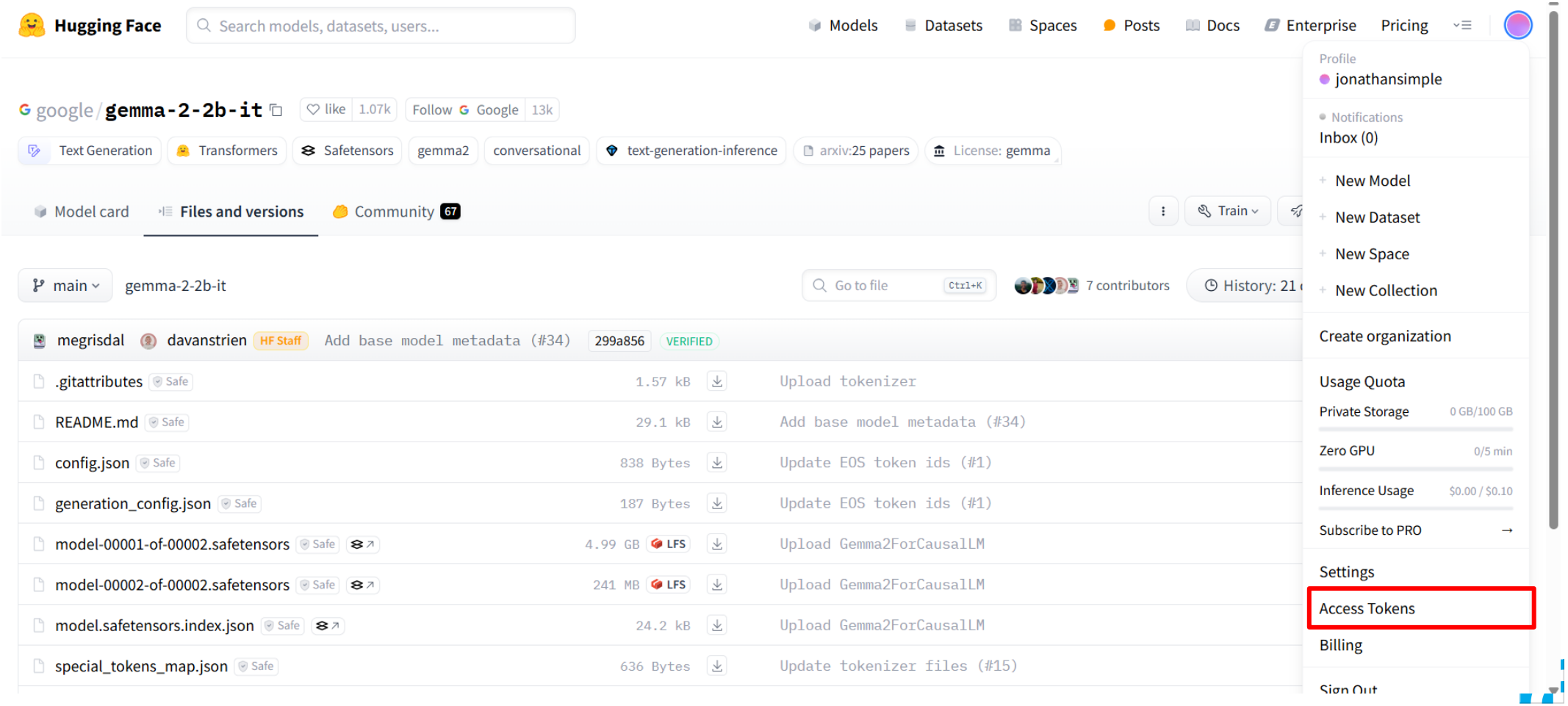
Huggingface Access Tokens

- Some models on Huggingface are gated, meaning you need permission to download them
  - Usually all this entails is accepting their license, some models are gated behind a paywall instead
- In such cases, you need to provide your Huggingface account's access tokens in order to download the model using methods 2 and 3
  - Manually downloading from the webpage while logged in bypasses the need to provide access tokens

©Industrial Technology Research Institute. All rights reserved.

How to Create Your Huggingface Access Tokens

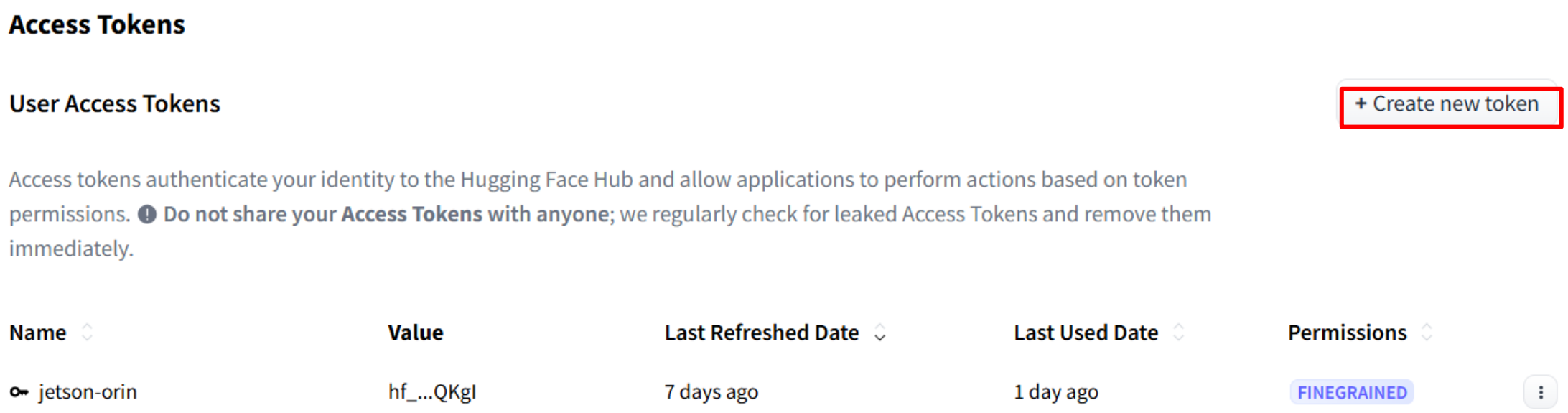
- Under account profile, find the “Access Tokens” category (you can also get there via Settings)



©Industrial Technology Research Institute. All rights reserved.

How to Create Your Huggingface Access Tokens

- Create your token with the desired read/write privilege and copy it for the next step



©Industrial Technology Research Institute. All rights reserved.

How to Provide Your Huggingface Access Tokens

Method 1: log into huggingface in your conda environment

- Install huggingface_hub with command line interface

```
pip install -U “huggingface_hub[cli]”
```

- Log in using your access token

```
huggingface-cli login
```

©Industrial Technology Research Institute. All rights reserved.

How to Provide Your Huggingface Access Tokens

Method 2: provide your access tokens when you call from_pretrained() [the function that pulls the model]

```
from transformers import AutoModel

access_token = "hf_..."

model = AutoModel.from_pretrained("private/model", token=access_token)
```

©Industrial Technology Research Institute. All rights reserved.

Transformers Example: Imports + Model Loading

```python
from transformers import AutoModelForCausalLM , AutoTokenizer
import os
import torch

os.environ["NCCL_P2P_DISABLE"] = "1"
os.environ["NCCL_IB_DISABLE"] = "1"

tokenizer = AutoTokenizer.from_pretrained("google/gemma-2-2b-it", trust_remote_code=True)
model = AutoModelForCausalLM.from_pretrained("google/gemma-2-2b-it", trust_remote_code=True, device_map="auto")

model.eval()
```

If your system doesn't support P2P GPU communication or InfiniBand (IB), set these environment variables otherwise your program may crash

Replace "google/…" with the path to the folder containing your LLM if you want to load your model + tokenizer from local file

Source code: inference_transformers.py

©Industrial Technology Research Institute. All rights reserved.

ITRI
Industrial Technology
Research Institute

LLM Deployment Using Transformers

©Industrial Technology Research Institute. All rights reserved.

Inferencing with Transformers

- Now that we have loaded our model, we can proceed to inferencing
- Five main steps:
  1. Generation configuration
  2. Query formatting
  3. Tokenize
  4. Generate/inference
  5. Decode

©Industrial Technology Research Institute. All rights reserved.

Inference Code Example (modified inference_transformers.py to fit on screen)

```python
from transformers import AutoModelForCausalLM , AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("google/gemma-2-2b-it", trust_remote_code=True)
model = AutoModelForCausalLM.from_pretrained("google/gemma-2-2b-it", trust_remote_code=True, device_map=0)
model.eval()

generation_config = dict(
    temperature=0.9,
    top_k=30,
    top_p=0.6,
    do_sample=True,
    num_beams=1,
    repetition_penalty=1.2,
    max_new_tokens=256
)

if __name__ == '__main__':

    for user_input in ["Who is the president of the United States?",
                       "Do you have any information on who the president of the United States is in 2025?"]:
        print("==> User input:", user_input)
        history_messages = [
            {"role": "user", "content": user_input},
        ]

        inputs = tokenizer.apply_chat_template(
            history_messages,
            add_special_tokens=False,
            add_generation_prompt=True,
            return_tensors="pt"
        )
        output = model.generate(
            input_ids = inputs.cuda(),
            **generation_config
        )[0]

        output = tokenizer.decode(output, skip_special_tokens=True)

        print("==> Gemma output:", output[output.find("model\n")+6:])
```

1. Set temperature to 0 (or do_sample to False) for deterministic outputs
1. Beam search setting: >1 for less randomness, ==1 for creativity
1. Sets the maximum length of the LLM's response
2. Generate a new chat session + specify whom (system/user/LLM) this string belongs to
3. Tokenize: convert chat log (string) to tokens
4. Inference step (output is tokens)
5. Convert output tokens to string

ITRI Industrial Technology Research Institute

©Industrial Technology Research Institute. All rights reserved.

Inference Code Example: Output (inference_transformers.py)

```
Loading checkpoint shards: 100%|                                                                    | 2/2 [00:02<00:00,  1.28s/it]
==> User input: Who is the president of the United States?
/home/sean/anaconda3/lib/python3.11/site-packages/torch/_inductor/compile_fx.py:150: UserWarning: TensorFloat32 tensor cores for float32 ma
trix multiplication available but not enabled. Consider setting `torch.set_float32_matmul_precision('high')` for better performance.
  warnings.warn(
==> Gemma output: The current President of the United States is **Joe Biden**.

==> User input: Do you have any information on who the president of the United States is in 2025?
==> Gemma output: It's impossible to know for sure!

Here's why:

* **The US Presidential election will take place in November 2024.** The winner of that election will be inaugurated as President in Januar
y 2025.
* **We don't yet know if a candidate will win or what their policies will be.** Political landscapes change rapidly, and anything can happe
n between now and then.


To stay up-to-date with potential candidates and political developments leading into the 2024 election, I recommend checking reputable news
 sources like:

* **Associated Press (AP)**
* **Reuters**
* **CNN**
* **Fox News**
* **New York Times**
* **Washington Post**



Let me know if you want more info about specific topics related to the upcoming elections!
```

This query exposes the older model’s outdated knowledge base

©Industrial Technology Research Institute. All rights reserved.

ITRI
Industrial Technology
Research Institute

LLM Deployment Alternatives

©Industrial Technology Research Institute. All rights reserved.

LLM Deployment Tools

Feature	Transformers	Ollama	LM Studio
Primary Use	Running, fine-tuning, and deploying LLMs	Local, easy-to-use deployment of LLMs	Local LLMs with GUI-based interaction
User Interface	Python code (CLI/Script-based)	CLI-based, simple for local models	GUI-based, user-friendly for local use
Hardware Requirements	Flexible (depends on model)	Light-weight, works on most systems	Local models (limited by hardware)
Deployment Scale	Small to large-scale (with proper setup)	Small to medium-scale (local use)	Personal or small-scale deployment
Target	Flexibility and ease of use for researchers	Simplicity and accessibility	Simplicity, local deployment
Performance	Slow	Fast	Fast

Out of all the deployment options, transformers is the **slowest**, transformers is mostly intended for development and not deployment

ITRI Industrial Technology Research Institute

©Industrial Technology Research Institute. All rights reserved.

Ollama

- Great tool for hosting LLMs locally
- Based on llama.cpp – optimized for speed
- Works on Linux, Windows, macOS
- Supports a wide variety of models – if you can’t find the model you are looking for, you can import your own model (in GGUF format)

ITRI
Industrial Technology Research Institute

©Industrial Technology Research Institute. All rights reserved.

Setting Up Ollama (Linux)

Installation:

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

Downloading a model:
ollama pull gemma3:1b-it-qat

Running a model:
ollama run gemma3:1b-it-qat

Print out all available (downloaded) models
ollama list

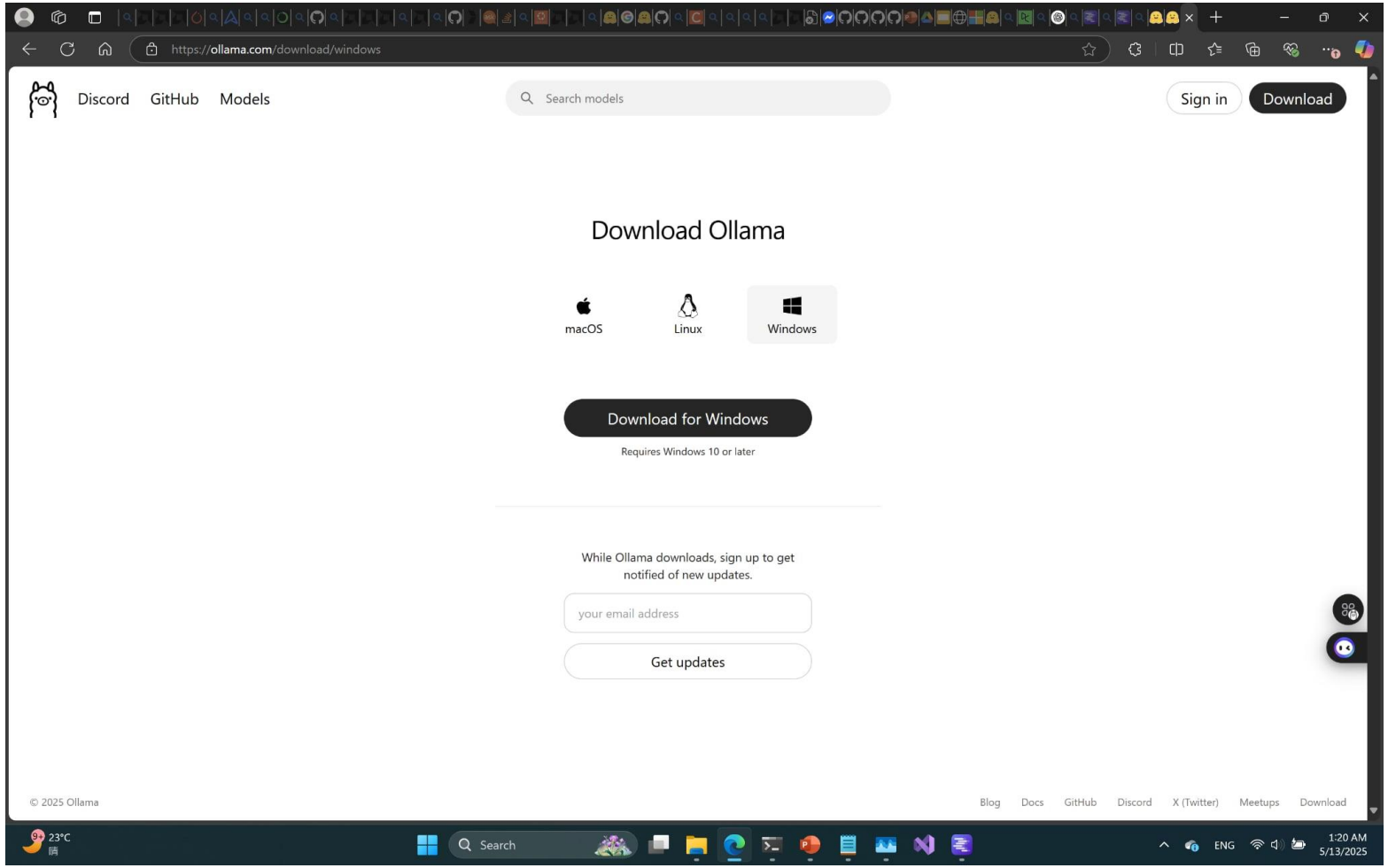
Model	Parameters	Size	Download
Gemma 3	1B	815MB	`ollama run gemma3:1b`
Gemma 3	4B	3.3GB	`ollama run gemma3`
Gemma 3	12B	8.1GB	`ollama run gemma3:12b`
Gemma 3	27B	17GB	`ollama run gemma3:27b`
QwQ	32B	20GB	`ollama run qwq`
DeepSeek-R1	7B	4.7GB	`ollama run deepseek-r1`
DeepSeek-R1	671B	404GB	`ollama run deepseek-r1:671b`
Llama 3.3	70B	43GB	`ollama run llama3.3`
Llama 3.2	3B	2.0GB	`ollama run llama3.2`
Llama 3.2	1B	1.3GB	`ollama run llama3.2:1b`
Llama 3.2 Vision	11B	7.9GB	`ollama run llama3.2-vision`
Llama 3.2 Vision	90B	55GB	`ollama run llama3.2-vision:90b`
Llama 3.1	8B	4.7GB	`ollama run llama3.1`

A non-exhaustive list of readily available models

ITRI Industrial Technology Research Institute

©Industrial Technology Research Institute. All rights reserved.

Setting Up Ollama (Windows)



A straightforward installer is available for Windows (simply download and run)

©Industrial Technology Research Institute. All rights reserved.

Running Ollama as a Chatbot on Command Line

```
PS C:\Users\jonas> ollama pull gemma2:2b
pulling manifest
pulling 7462734796d6: 100%                                                    1.6 GB
pulling e0a42594d802: 100%                                                     358 B
pulling 097a36493f71: 100%                                                    8.4 KB
pulling 2490e7468436: 100%                                                     65 B
pulling e18ad7af7efb: 100%                                                    487 B
verifying sha256 digest
writing manifest
success
PS C:\Users\jonas> ollama run gemma3:1b-it-qat
>>> Tell me about yourself
Hi there! I'm Gemma, a large language model created by the Gemma team at Google DeepMind. I'm designed to be an assistant that can
process and generate text.

Basically, I've been trained on a massive amount of text and code, which means I can do things like:

*   **Answer your questions:** I'll do my best to answer your questions thoroughly.
*   **Generate creative content:** I can write stories, poems, code, and more!
*   **Summarize text:** I can break down long pieces of text into shorter, more manageable summaries.
*   **Translate languages:** I can translate text from one language to another.
*   **And much more!**

I'm still under development, and I'm always learning new things.

How can I help you today?

>>> Send a message (/? for help)
```

Use the command: **ollama run** gemma3:1b-it-qat

©Industrial Technology Research Institute. All rights reserved.

Working With Modelfiles in Ollama (Linux)

With Modelfiles, you can have greater control over the LLM hosted on Ollama

Running your own model:
>>> ollama create mymodel –f modelfile.txt
>>> ollama run mymodel

Quantizing a model:
>>> ollama create --quantize q4_K_M mymodel_q –f modelfile.txt
>>> ollama run mymodel_q

```
FROM ./JD-llama.gguf
PARAMETER temperature 0.0
PARAMETER stop "<|im_start|>"
PARAMETER stop "<|im_end|>"
TEMPLATE """
<|im_start|>system
{{ .System }}<|im_end|>
<|im_start|>USER
{{ .Prompt }}<|im_end|>
<|im_start|>ASSISTANT
"""
```

An example of what a Modelfile would look like. The FROM line is mandatory (path to the model)

ITRI Industrial Technology Research Institute

©Industrial Technology Research Institute. All rights reserved.

Using Ollama as a Server

- A command line chatbot isn't very useful, but Ollama can be used as a server to handle query requests
- How do we set up an Ollama server?
  - On your machine's bootup, Ollama should already be running by default, if not, run `ollama serve`
- What models are available?
  - All of the model listed when you call `ollama list`
- What is the server's address?
  - By default, the Ollama server is posted on http://localhost:11434/v1

©Industrial Technology Research Institute. All rights reserved.

Sending a Query to Your Ollama Server (inference_ollama.py)

```
PS C:\Users\jonas> ollama list
NAME                ID              SIZE      MODIFIED
gemma2:2b           8ccf136fdd52    1.6 GB    8 hours ago
bge-m3:latest       790764642607    1.2 GB    9 hours ago
gemma3:1b-it-qat    b491bd3989c6    1.0 GB    10 hours ago
PS C:\Users\jonas>
```

List of available LLMs

```python
import requests
import openai


# ---------- CONFIG ----------
API_KEY = "sk-no-key-required" # Fake key (llama-server doesn't validate keys)

# Change this to your Ollama/LM Studio's host url or use localhost if running on the same machine
LLAMA_SERVER_URL = "http://localhost:11434/v1" # for ollama
MODEL_NAME = "gemma3:1b-it-qat" # the model identifier for the model you are using
```

The model name should match that shown in ollama list

©Industrial Technology Research Institute. All rights reserved.

Sending a Query to Your Ollama Server (inference_ollama.py)

```python
def inference(query_str: str):

    # prompt initialization
    conversations = [
        {"role": "system", "content": 'You are a helpful AI assistant named Bob.'},
        {"role": "user", "content": query_str},
    ]

    client = openai.OpenAI(base_url=LLAMA_SERVER_URL, api_key=API_KEY)

    # Stream the LLM output on the terminal
    stream = client.chat.completions.create(
        model=MODEL_NAME,
        messages=conversations,
        max_tokens=512,
        temperature=0.0,
        stream=True
    )

    llm_answer = ""
    print("Response:")

    for chunk in stream:
        if chunk.choices[0].delta.content:
            print(chunk.choices[0].delta.content, end="", flush=True)
            llm_answer += chunk.choices[0].delta.content
    print("\n")
    return llm_answer
```

Label your query and add any system prompts you wish

Create a client pointing to your Ollama server

The chat completion is where you specify which model you wish to use (notice how you are not explicitly calling tokenizers now, it's all being done beneath the hood)

Because we set stream=True, the LLM tokens will be streamed to the terminal in real time

ITRI Industrial Technology Research Institute

©Industrial Technology Research Institute. All rights reserved.

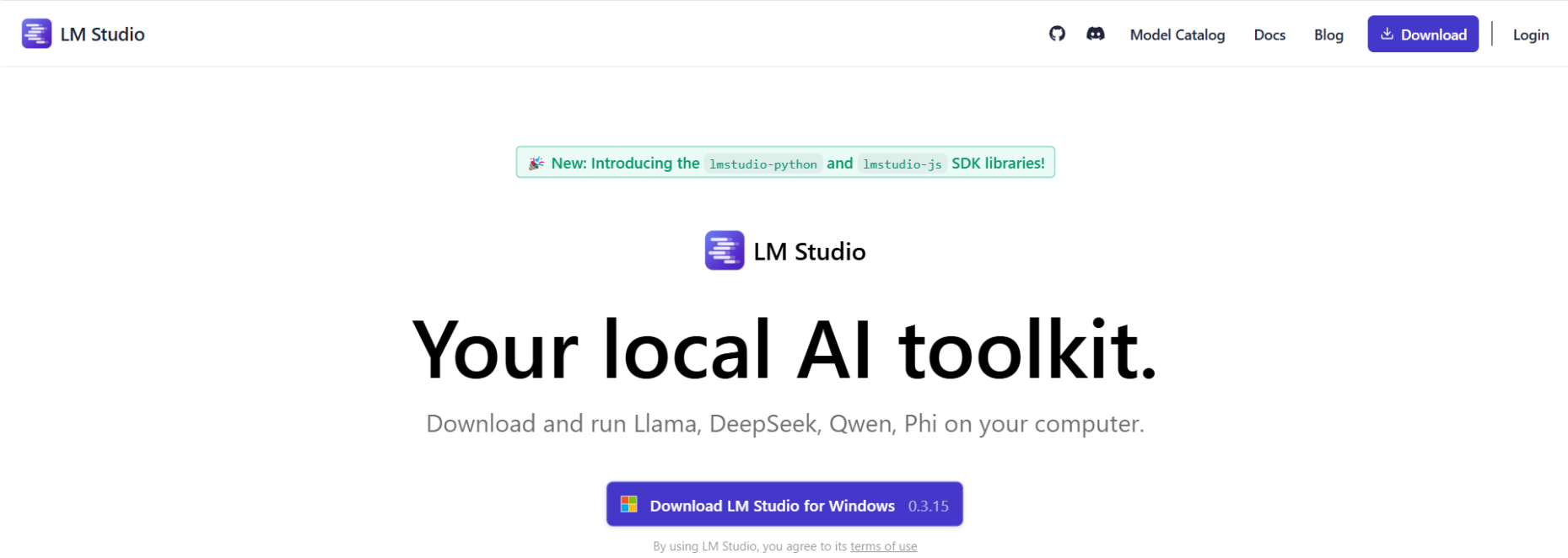
Using Ollama as a Server (inference_ollama.py)

```
(ryzen-ai) C:\Users\jonas\Desktop\workshop>python inference_ollama.py
How may I help you?
Hello, what's your name?
Response:
Hi there! I'm Bob! It's nice to meet you. 😊

--------------------------------------------------------------------------------
How may I help you?
Who is the president of the United States?
Response:
As of today, November 2, 2023, the President of the United States is Joe Biden.

Do you want to know anything else about him or the presidency in general?

--------------------------------------------------------------------------------
How may I help you?
```

©Industrial Technology Research Institute. All rights reserved.

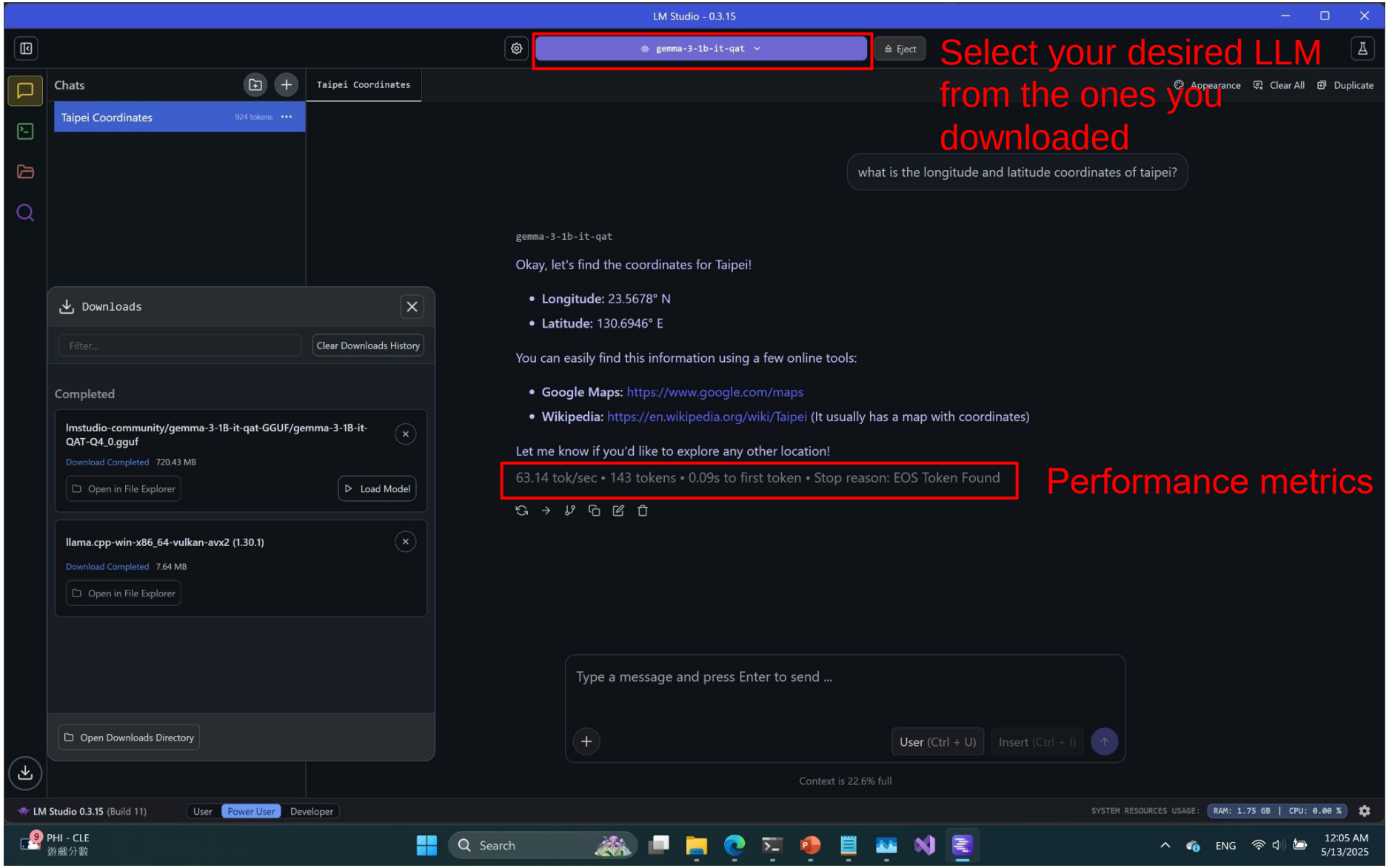
LM Studio (Windows)

- LM Studio is another tool for deploying LLMs that comes with a clean GUI
- LM Studio also has a lot of interesting stats readily available (token/s, total tokens, token generation termination cause)
- Great for beginners



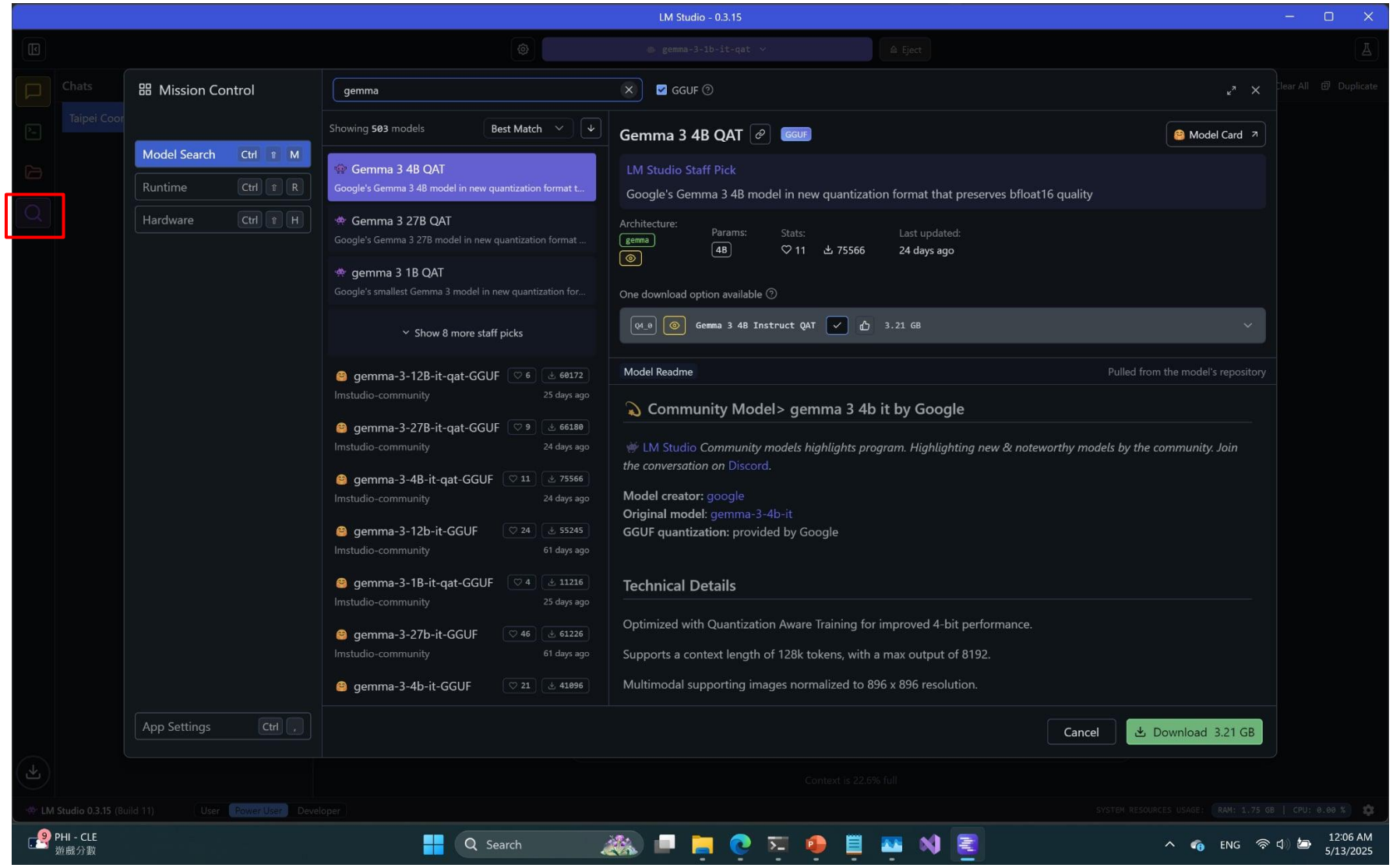
©Industrial Technology Research Institute. All rights reserved.

Running LM Studio as a Chatbot (Windows)



©Industrial Technology Research Institute. All rights reserved.

Downloading Models on LM Studio (Windows)

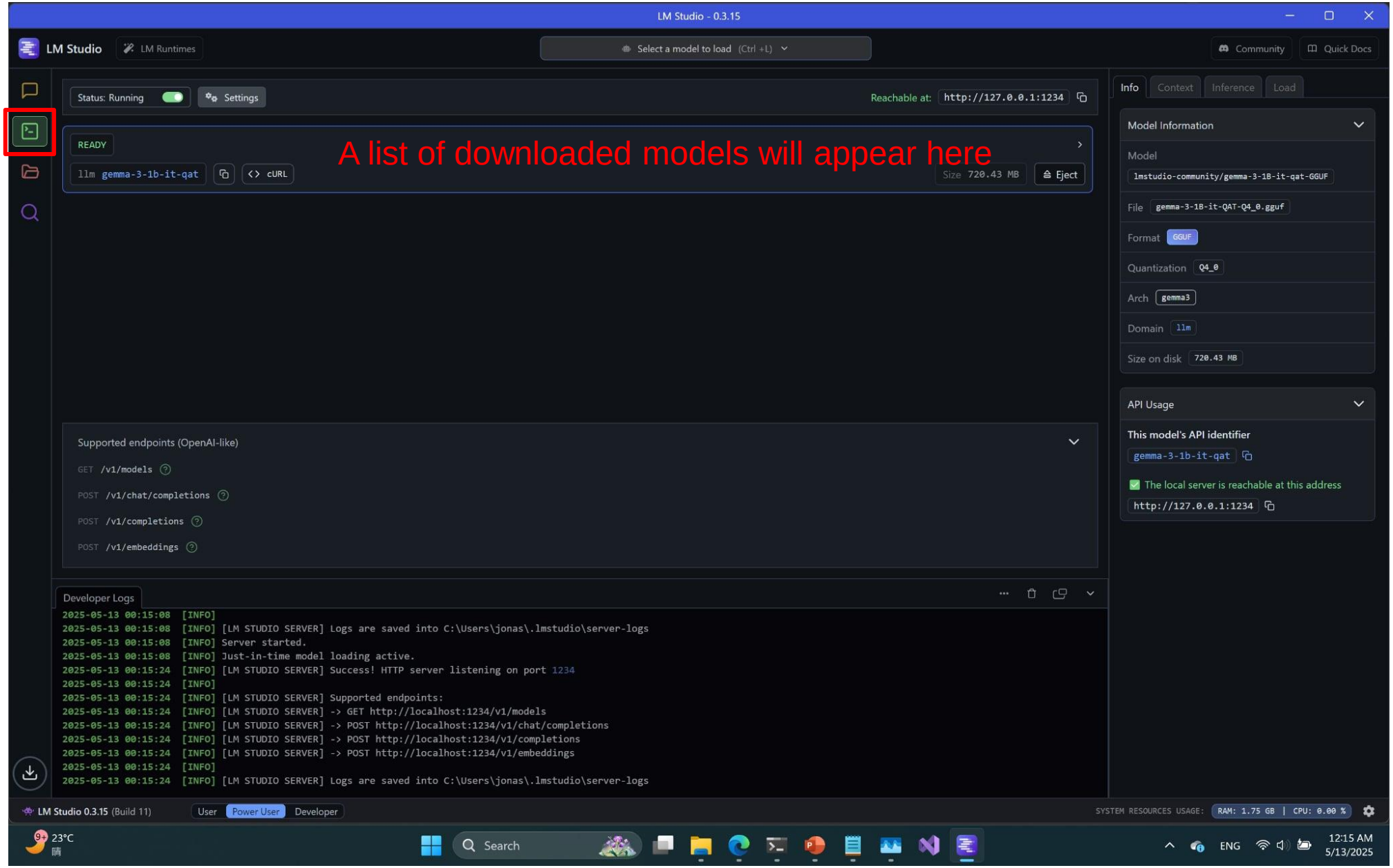


LM Studio can pull models off Hugging Face through the built in search sidebar

©Industrial Technology Research Institute. All rights reserved.

Using LM Studio to Host a Server

Go to the developer tab



©Industrial Technology Research Institute. All rights reserved.

Using LM Studio to Host a Server

Toggle the start server status to start your server

Developer Logs and API usage has useful information for setup and debugging

LM Studio - 0.3.15

LM Studio | LM Runtimes | Select a model to load (Ctrl +L) | Community | Quick Docs

Status: Running | Settings | Reachable at: http://127.0.0.1:1234

READY

llm gemma-3-1b-it-qat | cURL | Size 720.43 MB | Eject

Supported endpoints (OpenAI-like)

GET /v1/models

POST /v1/chat/completions

POST /v1/completions

POST /v1/embeddings

Developer Logs

```
2025-05-13 00:15:08  [INFO]
2025-05-13 00:15:08  [INFO] [LM STUDIO SERVER] Logs are saved into C:\Users\jonas\.lmstudio\server-logs
2025-05-13 00:15:08  [INFO] Server started.
2025-05-13 00:15:08  [INFO] Just-in-time model loading active.
2025-05-13 00:15:24  [INFO] [LM STUDIO SERVER] Success! HTTP server listening on port 1234
2025-05-13 00:15:24  [INFO]
2025-05-13 00:15:24  [INFO] [LM STUDIO SERVER] Supported endpoints:
2025-05-13 00:15:24  [INFO] [LM STUDIO SERVER] -> GET http://localhost:1234/v1/models
2025-05-13 00:15:24  [INFO] [LM STUDIO SERVER] -> POST http://localhost:1234/v1/chat/completions
2025-05-13 00:15:24  [INFO] [LM STUDIO SERVER] -> POST http://localhost:1234/v1/completions
2025-05-13 00:15:24  [INFO] [LM STUDIO SERVER] -> POST http://localhost:1234/v1/embeddings
2025-05-13 00:15:24  [INFO]
2025-05-13 00:15:24  [INFO] [LM STUDIO SERVER] Logs are saved into C:\Users\jonas\.lmstudio\server-logs
```

Info | Context | Inference | Load

Model Information

Model: lmstudio-community/gemma-3-1B-it-qat-GGUF

File: gemma-3-1B-it-QAT-Q4_0.gguf

Format: GGUF

Quantization: Q4_0

Arch: gemma3

Domain: llm

Size on disk: 720.43 MB

API Usage

This model's API identifier: gemma-3-1b-it-qat

The local server is reachable at this address: http://127.0.0.1:1234

LM Studio 0.3.15 (Build 11) | User | Power User | Developer | SYSTEM RESOURCES USAGE: RAM: 1.75 GB | CPU: 0.00 %

23°C 晴 | Search | ENG | 12:15 AM 5/13/2025

©Industrial Technology Research Institute. All rights reserved.

Using LM Studio to Host a Server

- Very similar to Ollama (in fact, code is **identical**, just change server address and model name)
- What is the server's address and model handle?
  - Check the API Usage sidebar for the server url and the model handle

©Industrial Technology Research Institute. All rights reserved.

ITRI
Industrial Technology
Research Institute

Fine Tuning with LoRA

©Industrial Technology Research Institute. All rights reserved.

How to Set Up LoRA Fine Tuning

- You'll need to install the peft library
- You can tune your LoRA hyperparameters using peft's LoraConfig()

```
# Load LoRA configuration
peft_config = LoraConfig(
    lora_alpha=32,        # Alpha parameter for LoRA scaling
    lora_dropout=0,     # Dropout probability for LoRA layers
    r=16,                 # LoRA attention dimension
    bias="none",
    task_type="CAUSAL_LM",
    target_modules=["q_proj", "k_proj", "v_proj", "o_proj",
                    "gate_proj", "up_proj", "down_proj",]
)
```

- During training, you pass this config to the trainer

```
# Set supervised fine-tuning parameters
trainer = SFTTrainer(
    model=model,
    train_dataset=dataset,
    peft_config=peft_config,
    args=training_arguments
)

trainer.train()
```

©Industrial Technology Research Institute. All rights reserved.

LoRA Hyperparameters

- For those just getting started with LoRA, focus on the r (rank) and alpha parameters
- If you run into CUDA OOM (out of memory) problems and you can’t reduce your batch size or max input length any further, you can try reducing the rank
- When starting out, it is recommended to maintain a 1:2 ratio between rank and alpha

©Industrial Technology Research Institute. All rights reserved.

Other Hyperparameters

- The trainer used for this demo comes from transformers' trl library (specifically SFTTrainer and SFTConfig)
- Here are some noteworthy parameters to tune while training your model:
  - max_seq_length: how many tokens your model can take in at once, affects your LLM's ability to handle long inputs; drastically affects VRAM usage
  - per_device_train_batch_size: batch size, adjust this based on your VRAM
  - save_strategy: how often you save checkpoints
  - num_train_epochs: how many epochs to train for (you can also opt to use steps as a unit instead of epoch)
  - learning_rate
  - bf16: whether to train in bf16
  - logging_steps: how often you log training statistics

ITRI Industrial Technology Research Institute

©Industrial Technology Research Institute. All rights reserved.

Example SFTConfig

```python
# Set supervised fine-tuning parameters
trainer = SFTTrainer(
    model=model,
    train_dataset=dataset,
    peft_config=peft_config,
    args=training_arguments
)

trainer.train()
```

```python
# Set training parameters
training_arguments = SFTConfig(
    output_dir="./results",
    overwrite_output_dir=True,
    save_strategy="no",
    num_train_epochs=1,
    per_device_train_batch_size=2,
    gradient_accumulation_steps=1,
    optim="adafactor",
    dataloader_drop_last=True,
    learning_rate=0.0002,
    bf16=True,
    max_grad_norm=0.3,
    warmup_ratio=0.03,
    lr_scheduler_type="linear",
    max_seq_length=1024,
    dataset_text_field="text",
    dataset_kwargs={
        "add_special_tokens": False,
        "append_concat_token": False,
    },
    packing=True,
    logging_steps=1,
    report_to="none",
    seed=42
)
```

See gemma_function_calling_finetune.py for a short example that you can run with a custom function calling dataset inside the ./dataset folder

©Industrial Technology Research Institute. All rights reserved.

ITRI
Industrial Technology
Research Institute

Inferencing - Function Calling

©Industrial Technology Research Institute. All rights reserved.

Function Calling Example: Function Declaration

```
system_prompt = r"""You are a helpful assistant with access to the following functions. Use them if required -
{
    "name": "get_weather",
    "description": "Get the current temperature for the provided coordinates in Celsius.",
    "parameters": {
        "type": "object",
        "properties": {
            "latitude": {"type": "number"},
            "longitude": {"type": "number"}
        },
        "required": ["latitude", "longitude"]
    }
}
If a function should be used, respond with the function name and its parameters in json format like so:
{"name": "function_name", "arguments": {"arg_1": "value_1", "arg_2": "value_2", ...}}

User Query: """
```

The **description** fields are important for informing the LLM on when a function should be used

```
User Query: Get me the weather in Taiwan.
model
```json
{"name": "get_weather", "arguments": {"latitude": 24.8375, "longitude": 120.6954}}
```
```

Sample output, the model provides the coordinates to Taiwan using its existing knowledge

©Industrial Technology Research Institute. All rights reserved.

Function Calling Example: API Integration

```python
import requests

def get_weather(latitude, longitude):
    response = requests.get(f"https://api.open-meteo.com/v1/forecast?latitude={latitude}&longitude={longitude}&current=temperature_2m")
    data = response.json()
    return data['current']['temperature_2m']

def parse_llm(llm_text):
    if r"```json" in llm_text:
        result = r"[" + llm_text.split(r"```json")[1].split(r"```")[0] + r"]"
    elif r"```tool_code" in llm_text:
        result = r"[" + llm_text.split(r"```tool_code")[1].split(r"```")[0] + r"]"
    elif r"<functioncall>" in llm_text:
        result = r"[" + llm_text.split(r"<functioncall>")[1].split(r"</functioncall>")[0] + r"]"

    print("==> Parsed result: ", result)

    try:
        data = json.loads(result)
        print("==> Parsed list:", data)

        return data
    except:
        print("Cannot be parsed")
        return None
```

API call

temperature_2m is the temperature 2 meters above ground

Handle LLM's structured output

Parse LLM output (string) into json

©Industrial Technology Research Institute. All rights reserved.

Function Calling Example: Inference Step (function_calling_transformers.py)

```python
def run_conversation(input_text):
    print("==> User Input:", input_text)

    history_messages = [
        {"role": "user", "content": system_prompt + input_text},
    ]

    inputs = tokenizer.apply_chat_template(
            history_messages,
            add_special_tokens=False,
            add_generation_prompt=True,
            return_tensors="pt"
    )

    # For CUDA or ROCm enabled devices
    if torch.cuda.is_available():
        inputs = inputs.cuda()

    start_time = time.time()
    function_calling_output = model.generate(
            input_ids = inputs,
            **generation_config
    )
    end_time = time.time()

    generated_tokens = function_calling_output.shape[-1] - inputs.shape[-1]
    elapsed_time = end_time - start_time
    tokens_per_sec = generated_tokens / elapsed_time

    function_calling_output = tokenizer.decode(function_calling_output[0],skip_special_tokens=True)

    print("==> Gemma output:", function_calling_output)

    data = parse_llm(function_calling_output)
    temperature = get_weather(data[0]["arguments"]["latitude"], data[0]["arguments"]["longitude"])
    print(f"The current temperature is {temperature} Celsius")
    print(f"Generated {generated_tokens} tokens in {elapsed_time:.2f}s → {tokens_per_sec:.2f} tokens/sec")
    print("-"*100)
```

Format user query and add function list to query

Tokenize string

Inference

Decode

Parse LLM output as json and call API

©Industrial Technology Research Institute. All rights reserved.

Function Calling Example: Inference Step (function_calling_ollama.py)

```
67  def run_conversation(input_text):
70      conversations = [
71          {"role": "user", "content": system_prompt + input_text},
72      ]
73
74      client = openai.OpenAI(base_url=LLAMA_SERVER_URL, api_key=API_KEY)
75
76      # Stream the LLM output on the terminal
77      stream = client.chat.completions.create(
78          model=MODEL_NAME,
79          messages=conversations,
80          max_tokens=512,
81          temperature=0.0,
82          stream=True
83      )
84
85      llm_answer = ""
86      print("==> Response:")
87
88      for chunk in stream:
89          if chunk.choices[0].delta.content:
90              print(chunk.choices[0].delta.content, end="", flush=True)
91              llm_answer += chunk.choices[0].delta.content
92      print("\n")
93
94      data = parse_llm(llm_answer)
95      temperature = get_weather(data[0]["arguments"]["latitude"], data[0]["arguments"]["longitude"])
96      print(f"The current temperature is {temperature} Celsius")
97      print("-"*100)
```

Format user query and add function list to query

Config client

Send request to server and stream output

Parse LLM output as json and call the correct function using function name and use the correct function input using arguments

Call API

©Industrial Technology Research Institute. All rights reserved.

Function Calling Example: Output (function_calling_ollama.py)

```
(ryzen-ai) C:\Users\jonas\Desktop\workshop>python function_calling_ollama.py
==> User Input: What is the weather in Taipei?
==> Response:
```json
{"name": "get_weather", "arguments": {"latitude": 25.06, "longitude": 121.50}}
```

==> Parsed result: [
{"name": "get_weather", "arguments": {"latitude": 25.06, "longitude": 121.50}}
]
==> Parsed list: [{'name': 'get_weather', 'arguments': {'latitude': 25.06, 'longitude': 121.5}}]
The current temperature is 31.7 Celsius

```

©Industrial Technology Research Institute. All rights reserved.

ITRI
Industrial Technology
Research Institute

RAG Using Ollama and LM Studio

©Industrial Technology Research Institute. All rights reserved.

Problem Rehash

- Why RAG?
  - Some models have outdated information, but we don't want to find tune the model
- Difference between RAG and normal inference:
  - Additional context for generation is provided (normally you only have the query)
  - As a result, your prompt will look different

- Here is a very simple RAG prompt:

Context: Donald Trump won the 2024 presidential election and remains president in 2025. Most new models can perform function calling out-of-the-box.

Question: Who is the president of the United States in 2025?

Answer:

©Industrial Technology Research Institute. All rights reserved.

Building a Vector Store for Retrieving Relevant Context (rag_query_ollama.py)

```python
# Example documents
documents = [
    "Ollama is way faster than transformers.",
    "Donald Trump won the 2024 presidential election and remains president in 2025.",
    "Most new models can perform function calling out-of-the-box."
]

# Embed documents using SentenceTransformer
embedder = SentenceTransformer('all-MiniLM-L6-v2')
print("Creating vector store...")
document_embeddings = embedder.encode(documents)

# Create a FAISS index
dimension = document_embeddings.shape[1]  # Dimension of embeddings
index = faiss.IndexFlatL2(dimension)
index.add(np.array(document_embeddings, dtype=np.float32))
```

One of many examples of building a vector store using FAISS

©Industrial Technology Research Institute. All rights reserved.

Generating a Response using LM Studio (rag_query_ollama.py)

```python
# Function to generate an answer using LM Studio's model
def generate_with_lm_studio(query, context):
    # prompt initialization
    conversations = [
        {"role": "system", "content": 'You are a helpful AI assistant named Bob capable of RAG (retrieval augmer
        {"role": "user", "content": f"Context: {context}\nQuestion: {query}\nAnswer:"},
    ]

    client = openai.OpenAI(base_url=LM_STUDIO_SERVER_URL, api_key=API_KEY)

    # Stream the LLM output on the terminal
    stream = client.chat.completions.create(
        model=MODEL_NAME,
        messages=conversations,
        max_tokens=512,
        temperature=0.0,
        stream=True
    )

    llm_answer = ""
    print("Response:")

    for chunk in stream:
        if chunk.choices[0].delta.content:
            print(chunk.choices[0].delta.content, end="", flush=True)
            llm_answer += chunk.choices[0].delta.content
    print("\n")

    return llm_answer
```

Very similar to the Ollama client code, the only difference here is that the system prompt and user prompt have been modified to be more RAG friendly.

©Industrial Technology Research Institute. All rights reserved.

Putting Everything Together (rag_query_ollama.py)

```python
def rag_system(query, documents, index, embedder):
    # Step 1: Perform retrieval (use FAISS for example)
    query_embedding = embedder.encode([query])[0]
    D, I = index.search(np.array([query_embedding], dtype=np.float32), k=2)
    retrieved_docs = [documents[i] for i in I[0]]

    # Step 2: Generate answer with LM Studio
    context = " ".join(retrieved_docs)
    answer = generate_with_lm_studio(query, context)
    return answer
```

Find the top 2 documents that most closely relate to the query

Concat the retrieved contexts for the RAG prompt

```python
# Example usage
query = "Who is the president of the United States in 2025?"
answer = rag_system(query, documents, index, embedder)
```

End-to-end function call

©Industrial Technology Research Institute. All rights reserved.

Conclusion

- In these slides, we discussed how to:
  1. Perform function calling using LLMs
  2. Improve a LLM's factual accuracy without fine tuning using RAG

- To do this, we covered:
  - What are LLMs, tokenizers, and function calling
  - How to craft prompts to maximize a LLM's potential
  - How to inference using a variety of platforms (transformers, Ollama, LM studio)
  - How to fine tune a LLM

ITRI
Industrial Technology Research Institute

©Industrial Technology Research Institute. All rights reserved.