Run ML Models Locally

lets deploy machine learning models locally and run them

what are the tools that are used to run a model locally

To deploy and run **machine learning models locally**, you need tools that handle:

- Model loading and inference
- (Optionally) model training
- Environment and hardware management
- Serving (if needed for API access)

🔧 Common Tools to Run ML Models Locally

Here’s a breakdown by category:

🧠 Model Runtimes (for inference)

Tool	Use Case	Notes
Ollama	Run LLMs (e.g., Mistral, LLaMA) with GGUF format	Simple CLI; GPU/CPU; supports quantized models
llama.cpp	Inference engine for LLMs in C++	Super lightweight; works well on CPU
ONNX Runtime	General-purpose ML inference	Works across platforms; GPU support; many pre-trained models
TensorFlow Lite	Lightweight inference (e.g., edge devices)	Optimized for mobile and IoT
PyTorch	Training + inference	Needs Python; common for custom models
Triton Inference Server (by NVIDIA)	Serve models over HTTP/gRPC	Works with PyTorch, TensorFlow, ONNX
Text Generation Web UI	Web UI for running LLMs	Works with llama.cpp, transformers, etc.
DeepSpeed	Accelerated inference for large models	Best with NVIDIA GPUs

---

🗃️ Model Formats

Format	Description	Supported By
GGUF	Quantized LLM format	llama.cpp, Ollama
ONNX	Cross-framework format	ONNX Runtime, Triton
TorchScript	Serialized PyTorch model	PyTorch

Format	Description	Supported By
SavedModel (TF)	TensorFlow model format	TensorFlow
MLIR / XLA	Lower-level formats for optimization	Advanced use cases

---

📦 Deployment/Environment Tools

Tool	Role
Docker	Containerized deployment
WSL2	Linux environment on Windows
Anaconda	Python env manager (esp. for ML)
Virtualenv / venv	Lightweight Python environments

🌐 Optional (Local Serving / UI)

Tool	Purpose
FastAPI / Flask	Wrap models into APIs
Gradio / Streamlit	Build simple UIs for local demos
Text Generation Web UI	Visual chat for LLMs

✅ Recommended Flow to Run a Model Locally

1. **Choose model** (e.g., LLaMA 3, Mistral, custom PyTorch/ONNX model)
2. **Pick runtime** based on format (Ollama, llama.cpp, ONNX Runtime)
3. **Install environment** (WSL, Docker, Anaconda)
4. **Run model** locally via CLI/UI/API
5. (Optional) Serve via Flask/Gradio for local access

Would you like a step-by-step example (e.g., running a Hugging Face ONNX model, or an LLM via Ollama)?

lets try
ollama
llama.cpp
pytorch
text generation web ui

explain what is wsl and how to install wsl2 to run these models and why wsl2 should be used later i'll give another prompt for further discussion

Great — you're picking four of the most practical tools to run models locally. Let's start by understanding **WSL2**, why you should use it, and how to install it.

🐧 What is WSL2?

WSL2 (Windows Subsystem for Linux v2) allows you to run a full **Linux distribution (e.g., Ubuntu)** inside **Windows**, without needing a separate dual-boot or virtual machine.

It gives native performance, **GPU support**, and **Docker compatibility**, making it ideal for running tools like:

- 🐍 `pytorch` (training/inference in Linux env)
- 🦙 `llama.cpp` (Linux C++ builds run faster and more natively)
- 🐳 Docker containers for Ollama and Web UI
- 🔧 Easier dependency management for ML tools

✅ Why Use WSL2 for ML Models?

Feature	WSL2 Advantage
Performance	Near-native speed with full Linux kernel
GPU Access	Supports NVIDIA GPU passthrough
Tool Support	Compatible with Linux-only ML tools
Docker	Docker Desktop uses WSL2 backend
File System Access	Share files between Windows & Linux easily

---

🔧 How to Install WSL2 (Step-by-Step)

🧱 Step 1: Enable WSL

Open **PowerShell as Administrator** and run:

```
powershell

wsl --install
```

This installs:

- WSL2 backend
- Ubuntu as the default distro
- Required components (VM Platform, Virtual Machine feature)

If it says WSL is already installed, force upgrade it:

```powershell
wsl --set-default-version 2
```

💻 Step 2: Install a Linux Distribution

Install **Ubuntu** from the Microsoft Store:

1. Open **Microsoft Store**
2. Search for **Ubuntu 22.04 LTS**
3. Click **Install**
4. Launch Ubuntu, set your username and password

⚙️ Step 3: Optional Upgrades

- ✅ Update packages:

```bash
sudo apt update && sudo apt upgrade -y
```

- ✅ Install essential tools:

```bash
sudo apt install build-essential git wget curl unzip -y
```

🐳 Step 4: Install Docker (Optional but recommended)

Install Docker inside WSL or use Docker Desktop (which uses WSL2 backend).

To install **Docker inside WSL2 Ubuntu**:

```bash
sudo apt install docker.io -y
sudo usermod -aG docker $USER
newgrp docker
```

Start Docker:

```bash
sudo service docker start
```

🧪 Verify WSL2 is Installed

In PowerShell:

```
powershell

wsl -l -v
```

You should see something like:

```
pgsql

NAME      STATE           VERSION
Ubuntu    Running         2
```

Once WSL2 is ready, you're good to:

- Compile llama.cpp models
- Run Ollama (inside Docker or directly)
- Use Python + PyTorch
- Run Text Generation Web UI

Would you like the next step: **installing and using each tool (ollama, llama.cpp, etc.) inside WSL2**?

will go with ollama first
explain how to install ollama with docker and without docker in wsl2 and then explain how to run models using ollama custom models and other ways to run models

explain each step as well
try to cover everything important regarding ollama here

Perfect. Let's dive deep into **Ollama** and cover everything from installation (with and without Docker in WSL2), to running both prebuilt and custom models.

🧠 What is Ollama?

Ollama is a tool that lets you easily run and chat with **local LLMs** like **Mistral, LLaMA**, and others. It:

- Uses the **GGUF format** (optimized for llama.cpp-based models)
- Supports **quantized models** (runs well even on CPUs)
- Provides a CLI and REST API for interaction
- Is simple, fast, and developer-friendly

⚙️ Installing Ollama in WSL2

You have two choices:

📦 Option 1: Install Ollama Without Docker (Recommended for Simplicity)

✅ Step-by-step:

1. **Open WSL2 (Ubuntu terminal)**
2. **Add Ollama's key and repo:**

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

This script:

- Downloads the latest version of `ollama`
- Installs it into `/usr/local/bin`
- Sets up required services

3. **Start the Ollama service:**

```bash
ollama serve
```

4. **Test it:**

```bash
ollama run mistral
```

First run will **download the model**, then you'll enter chat mode.

🐳 Option 2: Install Ollama With Docker in WSL2

⚠️ Not officially recommended by Ollama, but possible for containerization.

Steps:

1. **Install Docker inside WSL2** or use Docker Desktop with WSL2 integration.
2. **Pull the Ollama Docker image:**

```bash
docker pull ollama/ollama
```

3. **Run Ollama container:**

```bash
docker run -d --name ollama \
  --gpus all \
  -p 11434:11434 \
  -v ollama:/root/.ollama \
  ollama/ollama
```

4. **Check if it's running:**

```bash
curl http://localhost:11434
```

🚀 How to Run Models Using Ollama

🧪 A. Run a Prebuilt Model (e.g., Mistral)

```bash
ollama run mistral
```

You can also try:

```bash
ollama run llama2
ollama run codellama
ollama run gemma
```

This downloads the model if not present and opens chat.

⚙️ B. Run Custom GGUF Model from Hugging Face

1. Download GGUF model manually:

Visit a repo like:

```bash
https://huggingface.co/TheBloke/dolphin-2.6-mistral-GGUF
```

Right-click `.gguf` file → "Copy Link" → Use `wget`:

```bash
wget https://huggingface.co/TheBloke/dolphin-2.6-mistral-GGUF/resolve/main/dolphin-2.6-
mistral.Q4_K_M.gguf
```

Place it in:

```bash
~/.ollama/models
```

⚠️ If `~/.ollama/models` doesn't exist, create it manually.

2. Create a custom `Modelfile`

```Dockerfile
FROM llama2
PARAMETER temperature 0.7
PARAMETER top_p 0.9
```

Or specify local model path:

```Dockerfile
FROM ./dolphin-2.6-mistral.Q4_K_M.gguf
```

3. Build your model:

```bash
ollama create dolphin -f Modelfile
```

4. Run your model:

```bash
ollama run dolphin
```

🌐 C. Use REST API with Ollama

Start the server:

```bash
ollama serve
```

Make a request (example using `curl`):

```bash
curl http://localhost:11434/api/generate -d '{
  "model": "mistral",
  "prompt": "Explain quantum computing in simple terms"
}'
```

🛠️ Useful Commands

Command	Purpose
`ollama list`	Show downloaded models
`ollama pull <model>`	Download model manually
`ollama run <model>`	Run a model in chat
`ollama create <name> -f Modelfile`	Build custom model
`ollama serve`	Start REST API server
`ollama rm <model>`	Remove a model
`ollama help`	See all commands

📁 Ollama Model Storage

Models are stored in:

```
bash

~/.ollama/models
```

(If installed via Docker, it's stored in the `ollama` volume)

✅ Summary

Task	Method
Install	Script via WSL or Docker
Run model	`ollama run <model>`
Custom model	`ollama create <name> -f Modelfile`
REST API	`ollama serve` + curl/Postman

Would you like to proceed with downloading a specific GGUF model or try using Ollama inside a Python script/API next?

now do the same with llama.cpp

explain what it is, why it is important, how to install (no need of docker here), how to run models using llama.cpp

Great choice! Let’s now cover **llama.cpp** — a powerful, lightweight tool for running LLMs **locally** with **no cloud dependency**, perfect for WSL2 setups.

🦙 What is `llama.cpp`?

`llama.cpp` is a C++ implementation of Meta's LLaMA models (and compatible models like Mistral, Gemma, etc.) that:

- Runs on **CPU and GPU (CUDA, Metal, OpenCL, Vulkan, etc.)**
- Supports **quantized GGUF models** (efficient, small-size models)
- Offers **low system requirements** (can run 7B models on 8GB RAM)
- Is **fully offline**
- Is **used by tools like Ollama, LM Studio, and Text Generation Web UI**

✅ Why Use `llama.cpp`?

Advantage	Reason
✅ Portable	Single binary, no Python needed
✅ Efficient	Works on low-spec machines
✅ Customizable	Full control via CLI or API
✅ GGUF Support	GGUF is optimized for CPU/GPU inference
✅ Open Source	MIT licensed, hackable

⚙️ How to Install `llama.cpp` in WSL2 (No Docker Needed)

🧰 Prerequisites

Run this in your **WSL2 Ubuntu terminal**:

```bash
sudo apt update && sudo apt install build-essential cmake git -y
```

🔧 Step-by-Step Installation

1. **Clone the repo:**

```bash
git clone https://github.com/ggerganov/llama.cpp
cd llama.cpp
```

2. **Build the project:**

```bash
make
```

This creates the `main` binary in the same folder. You now have a working CLI tool.

✅ Optional: Add it to your `PATH` if you want to run from anywhere.

📦 How to Download a GGUF Model

Let's say you want to use **Mistral 7B** or **Dolphin-Mistral** from Hugging Face.

1. Visit a model repo like:

```arduino
https://huggingface.co/TheBloke/Mistral-7B-Instruct-v0.2-GGUF
```

2. Right-click a file (e.g., `mistral-7b-instruct-v0.2.Q4_K_M.gguf`) → Copy Link
3. Download it in WSL:

```bash
wget https://huggingface.co/TheBloke/Mistral-7B-Instruct-v0.2-GGUF/resolve/main/mistral-7b-instruct-v0.2.Q4_K_M.gguf -O mistral.gguf
```

🚀 How to Run a Model with llama.cpp

Basic command (after build):

```bash
./main -m mistral.gguf -p "What is artificial intelligence?"
```

🔁 Continuous chat mode:

```bash
./main -m mistral.gguf -i
```

It keeps the session open — just like a chatbot.

🧠 Advanced Parameters (Optional)

Flag	Description
`-t 4`	Use 4 threads (adjust for CPU)
`-n 256`	Limit output to 256 tokens
`-p`	Prompt input
`-i`	Interactive chat mode

Flag	Description
`--color`	Adds color to output
`--temp 0.7`	Set temperature
`--top-p 0.95`	Top-p sampling

Example:

```bash
./main -m mistral.gguf -t 4 -n 256 --temp 0.7 --top-p 0.9 -i
```

🧪 Optional: GPU Support

You can compile with CUDA (NVIDIA GPU):

```bash
make clean
LLAMA_CUBLAS=1 make
```

For Metal (macOS) or OpenCL (Intel/AMD), use appropriate flags from the README:
https://github.com/ggerganov/llama.cpp#build

📂 Organizing Models

Keep your GGUF models in a `models/` folder:

```bash
mkdir models
mv mistral.gguf models/
```

Then run:

```bash
./main -m models/mistral.gguf -i
```

🛠️ Useful Files and Tools in llama.cpp

File	Purpose
`main`	CLI tool for text generation
`quantize`	Quantize original FP16 models to GGUF
`convert.py`	Convert HF models to GGUF (advanced users)
`server`	Run as API (basic local server)
`chat`	Interactive CLI (more chatbot-like)

📌 Summary

Step	Action
Clone & Build	**`git clone && make`**
Get Model	Download **`.gguf`** from Hugging Face
Run Model	**`./main -m model.gguf -i`**
Customize	Use flags for threads, temp, tokens
GPU Use	Rebuild with **`LLAMA_CUBLAS=1`**

Would you like to proceed with Text Generation Web UI or want a step-by-step demo using a specific model (e.g., running Dolphin in llama.cpp)?

lets continue with pytorch
explain everything like above

Absolutely! Now let's cover **PyTorch** — a powerful and flexible ML framework widely used for developing, training, and running machine learning models, including LLMs and custom deep learning models.

🔥 What is PyTorch?

PyTorch is an open-source deep learning framework developed by Meta. It is known for:

- **Dynamic computation graph** (eager execution)
- Extensive model library (vision, NLP, LLMs)
- Deep integration with **Hugging Face**, **transformers**, and **TorchScript**
- GPU support via **CUDA**
- Powerful for both **research and production**

✅ Why Use PyTorch?

Feature	Why It Matters
🧠 Research friendly	Code feels like native Python
⚙️ Versatile	From LLMs to image classification
💻 GPU support	Fast training/inference
🤝 Ecosystem	Works with Hugging Face, torchvision, torchaudio, etc.
🌐 Exportable	Can be exported to ONNX, TorchScript for deployment

---

⚙️ How to Install PyTorch in WSL2 (CPU or GPU)

🐍 Recommended: Use Python Virtual Environment

```
bash

sudo apt update && sudo apt install python3-venv python3-pip -y
python3 -m venv pytorch-env
source pytorch-env/bin/activate
```

💻 Step 1: CPU-only Installation (lightweight)

```
bash

pip install torch torchvision torchaudio
```

⚡ Step 2: GPU-enabled Installation (CUDA support)

Only if your WSL2 has **NVIDIA GPU** with CUDA support:

```
bash

pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu121
```

> You can change `cu121` to match the CUDA version you have.

📦 How to Load and Run a Model in PyTorch

Here's a basic example with a **pretrained ResNet** model for image classification:

```
python

import torch
from torchvision import models, transforms
from PIL import Image

# Load model
model = models.resnet18(pretrained=True)
model.eval()

# Load and preprocess image
image = Image.open("example.jpg")
preprocess = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor()
])
input_tensor = preprocess(image).unsqueeze(0)

# Run inference
with torch.no_grad():
    output = model(input_tensor)

print("Predicted class index:", output.argmax().item())
```

🧠 Run a Text Model (Hugging Face + PyTorch)

Install Transformers:

```
bash

pip install transformers
```

Run a BERT model:

```
python

from transformers import pipeline

classifier = pipeline("sentiment-analysis")
result = classifier("PyTorch makes AI easy!")
print(result)
```

🧪 Training Your Own Model (Minimal Example)

```
python

import torch
import torch.nn as nn
import torch.optim as optim

# Dummy data
X = torch.rand(100, 1)
y = 3 * X + 2 + 0.1 * torch.randn(100, 1)

# Model
model = nn.Linear(1, 1)
loss_fn = nn.MSELoss()
optimizer = optim.SGD(model.parameters(), lr=0.01)

# Training loop
for epoch in range(200):
    pred = model(X)
    loss = loss_fn(pred, y)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

print("Learned weights:", model.weight.item(), model.bias.item())
```

🗃️ TorchScript: Export Models

Save a trained model:

```
python

torch.save(model.state_dict(), "model.pt")
```

Load and run later:

```python
python

model.load_state_dict(torch.load("model.pt"))
model.eval()
```

🧩 Model Sources

Platform	Notes
🤗 Hugging Face	Thousands of PyTorch-compatible models
torchvision	Vision models
torchaudio	Audio models
custom	Build your own (from scratch or fine-tune)

📌 Summary

Task	Command/Tool
Install PyTorch	**`pip install torch torchvision`**
GPU Support	Use CUDA index URL
Run pre-trained models	Use torchvision or transformers
Train your own	Use **`nn.Module`**, **`optim`**, etc.
Save/Load	**`torch.save`**, **`torch.load`**
Export for deployment	TorchScript or ONNX

Let me know if you want:

- A full local chatbot using PyTorch + transformers
- Running PyTorch inside Flask or Gradio
- Or should we now move to **Text Generation Web UI**?

now explain the same for text generation web ui
same above things

Perfect! Let's now cover **Text Generation Web UI**, which acts as a powerful **graphical interface** to run and interact with large language models like LLaMA, Mistral, GPT-J, Falcon, and more — locally.

🌐 What is Text Generation Web UI?

Text Generation Web UI is a locally hosted, browser-based interface built with Gradio that lets you:

- Load and chat with LLMs (GGUF, Transformers, etc.)

- Use llama.cpp, GPTQ, ExLlama, AutoGPTQ, Transformers, etc.
- Run models **fully offline**
- View token-by-token output with adjustable sampling parameters
- Extend features with extensions like character chat, API access, etc.

It supports models in **GGUF, GGML, GPTQ, ExLlama, Hugging Face format**, and more.

✅ Why Use Text Generation Web UI?

Feature	Reason
🧑‍💻 User-Friendly	GUI makes it easy for non-technical users
🔁 Model Flexibility	Supports many model formats (GGUF, GPTQ, HF Transformers)
🧩 Extensions	Character AI, LoRA, APIs, etc.
🧠 Backend Options	llama.cpp, AutoGPTQ, Transformers, ExLlama, etc.
🖥️ Local	100% offline support with WSL2
⚙️ Highly Configurable	Advanced prompt control, sampling, etc.

⚙️ How to Install in WSL2 (CPU/GPU – No Docker)

🧰 Step 1: Prepare Python Environment

```
bash

sudo apt update && sudo apt install git python3-venv python3-pip -y
```

Create and activate virtual environment:

```
bash

python3 -m venv tg-ui-env
source tg-ui-env/bin/activate
```

📥 Step 2: Clone the Repository

```
bash

git clone https://github.com/oobabooga/text-generation-webui.git
cd text-generation-webui
```

🔧 Step 3: Install Requirements

Install minimal base:

```bash
pip install -r requirements.txt
```

If you plan to use Hugging Face transformers (PyTorch), also install:

```bash
pip install transformers accelerate
```

For `llama.cpp` models (GGUF), install:

```bash
pip install ctransformers
```

🚀 Step 4: Run the Web UI

Start the UI (for now, use basic Transformers backend):

```bash
python server.py
```

Then open in browser:
`http://localhost:7860`

📦 How to Load Models (GGUF or Hugging Face)

Option A: GGUF (llama.cpp backend)

1. **Download GGUF model** (e.g., Mistral from TheBloke):

```bash
mkdir -p models/llama.cpp
cd models/llama.cpp
wget https://huggingface.co/TheBloke/Mistral-7B-Instruct-v0.2-
GGUF/resolve/main/mistral.Q4_K_M.gguf -O mistral.gguf
```

2. Run with llama.cpp loader:

```bash
python server.py --model mistral.gguf --loader llama.cpp
```

Option B: Transformers Model (e.g., GPT2, Falcon, LLaMA2)

```
bash

python server.py --model TheBloke/Mistral-7B-Instruct-v0.2
```

Optional flags:

- **`--trust-remote-code`** (needed for some HF models)
- **`--chat`** (for chatbot UI)
- **`--loader transformers`** (explicitly use Hugging Face backend)

⚙️ Extensions and Features

Enable extensions by adding **`--extensions`** flag:

```
bash

python server.py --extensions openai,gallery,superbooga
```

Available extensions include:

- **`openai`** — serve API like OpenAI
- **`character`** — character-based chat
- **`gallery`** — prompt templates
- **`superbooga`** — multi-model switching UI

💡 Useful Options

Flag	Description
`--chat`	Enable chat mode
`--model`	Load specific model
`--loader`	Select backend: llama.cpp, transformers, etc.
`--listen`	Allow access from other devices
`--extensions`	Enable extensions
`--auto-devices`	Auto-select available GPUs
`--load-in-8bit`	Load model in 8-bit mode to save RAM

📁 Folder Structure

Path	Purpose
`models/`	Place downloaded models here
`text-generation-webui/`	Main repo
`extensions/`	Optional plugins

🧪 Sample Use Cases

- Chat with LLaMA or Mistral models offline via GUI
- Serve your own chatbot API from localhost
- Fine-tune prompt engineering using temperature, top-p, etc.
- Test GGUF vs Transformers model performance side-by-side

📌 Summary

Task	Command/Tool
Install	`git clone`, `pip install -r requirements.txt`
Run UI	`python server.py`
Load GGUF	`--loader llama.cpp`
Load Transformers	`--loader transformers`
Use API	`--extensions openai`
Browse	`http://localhost:7860`

Would you like:

- A comparison chart between Ollama vs Text Gen UI vs llama.cpp?
- A script to automate model download + running with the UI?
- Or guide to run **multiple models and switch between them** inside the UI?