

Using LLMs on “Aristotelis” HPC infrastructure

George Vlahavas
Researcher @ Datalab, AUTH

Outline

- LLMs: Open source vs Proprietary
- How to run LLMs locally with Ollama
- How to run LLMs on Aristotelis HPC infrastructure
- Performance comparisons
- Comparison of different models: from small to large

LLMs: Open source vs Proprietary

- LLMs:
Open source (local deployment)
vs. Proprietary (e.g., ChatGPT)
- Pros & Cons
- Requirements for local LLMs

Proprietary LLMs: Pros & Cons

- **Pros**

- **No need for infrastructure:** provided as a service
- **Ease of use:** available UIs, APIs
- **Accuracy:** trained with tons of data
- **Scalability:** can handle high volumes of data, suitable for enterprise-level applications

- **Cons**

- **Privacy:** Data (e.g., prompts, text for annotation) provided in proprietary LLM services (e.g., ChatGPT) become available at the company (e.g., OpenAI).
- **Cost:** paid subscription for access, cost per API request
- **Dependence on provider:** no access if the provider experiences issues
- **Usage limits:** rate limits, token limits, monthly limits, etc
- **Censored models:** will not respond to *any* prompt

Open-source LLMs (local deployment): Pros & Cons

- **Pros**

- **Privacy:** Information & data stay local (at your system)
- **Cost:** zero cost for usage
- **Usage limits:** no limits for usage (apart from local hardware limitations)
- **Uncensored models:** there are versions of the more popular models that will reply to any prompt
- **Offline:** once you have downloaded a model, you don't need a network connection
- **Customizability:** users can modify the models to their specific needs
- **Enhanced security:** can be audited for security vulnerabilities

- **Cons**

- **Need for infrastructure:** Infrastructure (typically powerful) is needed to support, in particular, large LLMs
→ ...*but, see “Aristotelis” HPC infrastructure*
- **Ease of use:** need to deploy & run service
→ ... *but, availability UIs/APIs availability & today's how-to manual*
- **Accuracy:** typically much smaller models
→ ...*but, things are progressing quite fast and fine tuning is possible*

Requirements for Local LLMs

Model Parameters	RAM	HD Space
2B	4 GB	~1.5 GB
7B	8 GB	~4.5 GB
13B	16 GB	~7.5 GB
33B	32 GB	~20 GB
70B	64 GB	~40 GB

RAM can be VRAM or system RAM. Ollama will offload as many layers as can fit to the GPU and process the remaining with system RAM/CPUs.

How to run LLMs locally with *Ollama*

- The Ollama framework
- Installation & usage
- Local hardware limitations

What is Ollama?

- Ollama is a software that allows users to build and run LLMs locally. Its functionality is comparable to Docker, but for LLMs.
- It can be used in many ways: interactive shell, API, Python library...
- It contains a library of hundreds of pre-built models that can be easily used in a variety of applications, including **LLaMA3**, **Qwen**, **Mistral** and **Phi4**
- Will use a GPU if there is one, otherwise will fallback to CPU
- Website: <https://ollama.com/>
- Github: <https://github.com/ollama/ollama>

Using Ollama locally

- Download and install according to the instructions at <https://ollama.com/download>
- There are versions for:
 - Linux
 - macOS
 - Windows



Get up and running with large language models.

Run [Llama 2](#), [Code Llama](#), and other models.
Customize and create your own.

Download ↓

Available for macOS, Linux,
and Windows (preview)

Using Ollama locally - starting the service

- The Ollama service provides a locally deployed service that ollama clients can connect to
- To start the Ollama service, you can launch the desktop app (macOS/Windows)
- Or start the service from the command line (all platforms):

\$ ollama serve

```
$ ollama serve
...
time=2025-01-13T15:05:38.267+02:00 level=INFO source=routes.go:1310 msg="Listening on 127.0.0.1:11434 (version 0.5.4)"
...
```

Using Ollama locally - managing models

- Download a model with:

```
$ ollama pull <model_name>
```

Example:

```
$ ollama pull llama3.2:1b
```

- Get a list of all models you have downloaded:

```
$ ollama list
```

- Remove a model that you no longer want:

```
$ ollama rm <model_name>
```

```
$ ollama list
NAME                ID                SIZE    MODIFIED
qwen2.5:32b         9f13ba1299af     19 GB   12 minutes ago
llama3.3:70b        a6eb4748fd29     42 GB   About an hour ago
qwen2.5:14b         7cdf5a0187d5     9.0 GB  2 hours ago
phi4:14b            ac896e5b8b34     9.1 GB  2 hours ago
qwen2.5:7b          845bdba0ea48     4.7 GB  2 hours ago
gemma:2b            b50d6c999e59     1.7 GB  3 hours ago
gemma:7b            a72c7f4d0a15     5.0 GB  3 hours ago
llama3.2:3b         a80c4f17acd5     2.0 GB  3 hours ago
llama3.1:8b         46e0c10c039e     4.9 GB  4 hours ago
llama3.2:1b         baf6a787fdff     1.3 GB  4 hours ago
llama3.2-vision:11b 085a1fdae525     7.9 GB  4 hours ago
qwen2.5-coder:7b    2b0496514337     4.7 GB  4 hours ago
```

Using Ollama locally - running a model

- You can interact with a model by running:

```
$ ollama run <model_name>
```

- When using the run command, if a non-local model is selected, it will be downloaded automatically

```
$ ollama run mistral:7b
pulling manifest
pulling ff82381e2bea... 49%  | 2.0 GB/4.1GB 5.2 MB/s 6m45s
```

Using Ollama locally - interaction

- After running `$ ollama run <model_name>` you get a message prompt:

```
$ ollama run llama3.1:8b
>>> Send a message (/? for help)
```

- You can then type any prompt and get an answer from your desired model

```
$ ollama run llama3.1:8b
>>> What university has the initials AUTH?
The university with the initials AUTH is Aristotle University of Thessaloniki, located in Greece.
>>> Send a message (/? for help)
```

- To exit type: `/bye` (or Ctrl-D)

```
>>> /bye
$
```

Using Ollama locally - available models



You can get a full list of available models in the ollama homepage:

<https://ollama.com/models>

The screenshot shows the Ollama models page interface. At the top is a search bar labeled "Search models". Below it are filter tabs: "All" (selected), "Embedding", "Vision", and "Tools". On the right is a "Popular" dropdown menu. The first model listed is "llama3.3", described as a "New state of the art 70B model. Llama 3.3 70B offers similar performance compared to the Llama 3.1 405B model." It has tags for "tools" and "70b", and statistics: 746.9K Pulls, 14 Tags, and Updated 5 weeks ago. The second model is "phi4", described as a "14B parameter, state-of-the-art open model from Microsoft." It has a "14b" tag and statistics: 118.8K Pulls, 5 Tags, and Updated 8 days ago. The third model is "qwq", described as an "experimental research model focused on advancing AI".

Local Hardware Limitations

- Trying to run local LLMs on:
 - A desktop PC:
 - Intel Core i7-9700K CPU @ 3.60GHz
 - 32 GB RAM
 - NVIDIA GeForce GTX 1050 Ti 4GB
 - A laptop:
 - Intel Core i7-1165G7 @ 2.80GHz
 - 32 GB RAM
 - No GPU

	Desktop		Laptop
Model	Layers in GPU	Tokens/s	Tokens/s
llama3.2:1b	17/17	49	17
llama3.2:3b	29/29	26	11
qwen2.5:7b	19/29	8.7	4.4
llama3.1:8b	20/33	7.8	4.0
qwen2.5:14b	14/49	3.4	2.1
qwen2.5:32b	7/65	1.4	?
llama3.3:70b	ERROR	-	?

How to run LLMs on Aristotelis HPC infrastructure

- Available Partitions
- Ollama Installation
- OnDemand
- Batch jobs
- Interactive session
- REST API
- Jupyter notebook
- Clients and UIs
- Potential Issues

How to run LLMs on Aristotelis HPC infrastructure

- Available Partitions
- Ollama Installation
- OnDemand
- Batch jobs
- Interactive session
- REST API
- Jupyter notebook
- Clients and UIs
- Potential Issues

Available Partitions

Partition	Nodes	GPU	VRAM Size
ampere	1	8x NVIDIA A100	40 GB
gpu	2	NVIDIA Tesla P100	12 GB
ondemand	12	NVIDIA Quadro RTX 6000	6 GB
a4000	1	NVIDIA RTX A4000	16 GB
rome	17	-	-
batch	20	-	-

More info on <https://hpc.it.auth.gr/nodes-summary/>

and <https://hackmd.io/@pkoro/HytY4qQWA#/>

How to run LLMs on Aristotelis HPC infrastructure

- Available Partitions
- **Ollama Installation**
- OnDemand
- Batch jobs
- Interactive session
- REST API
- Jupyter notebook
- Clients and UIs
- Potential Issues

Ollama Installation on Aristotelis

- Use preinstalled module (easiest but may lag a few versions behind)

```
$ module load ollama
```

- Install in user's home (latest version)

```
$ # install ollama, run these whenever there is a new release available  
$ mkdir -p $HOME/ollama  
$ cd $HOME/ollama  
$ rm -rf lib bin ollama-linux-amd64.tgz  
$ wget https://ollama.com/download/ollama-linux-amd64.tgz  
$ tar xf ollama-linux-amd64.tgz  
$ # to use it, just add it to your $PATH  
$ export PATH=$PATH:$HOME/ollama/bin
```

How to run LLMs on Aristotelis HPC infrastructure

- Available Partitions
- Ollama Installation
- **OnDemand**
- Batch jobs
- Interactive session
- REST API
- Jupyter notebook
- Clients and UIs
- Potential Issues

OnDemand

- Running OnDemand is just like running locally
- OnDemand nodes offer Nvidia Quadro RTX 6000 6GB GPUs
- Follow the instructions at <https://hpc.it.auth.gr/web-portal/> to login to Aristotle Desktop
- Run **ollama serve** in one terminal
- Run **ollama run <model_name>** on another

OnDemand - Potential Issue

- If someone else is already running an ollama service on the same node, you'll get an error message
- Solution: specify a different port, try until it works

```
$ ollama serve
Error: listen tcp 127.0.0.1:11434: bind: address already in use
$ export OLLAMA_HOST=127.0.0.1:15678
$ ollama serve
```

- Remember to export **OLLAMA_HOST** when running **ollama run** too.

```
$ export OLLAMA_HOST=127.0.0.1:15678
$ ollama run llama3.2:1b
```

How to run LLMs on Aristotelis HPC infrastructure

- Available Partitions
- Ollama Installation
- OnDemand
- **Batch jobs**
- Interactive session
- REST API
- Jupyter notebook
- Clients and UIs
- Potential Issues

First Steps

- You need to login with `ssh` (<https://hpc.it.auth.gr/intro/>) to a login node. Run:

```
$ ssh <username>@aristotle.it.auth.gr
```
- You need a VPN if you are not inside AUTH network. VPN instructions here: <https://it.auth.gr/manuals/eduvpn/>
- You can also use the Web Portal to login (<https://hpc.it.auth.gr/web-portal/>). No VPN needed.
- Then you need the bash script to submit your job...
- Instructions about submitting ollama batch scripts at <https://hpc.it.auth.gr/applications/ollama/>

Batch job

Create SLURM submission script and use Ollama through that

```
#!/bin/bash
#SBATCH --job-name=ollama-batch
#SBATCH --partition=ampere
#SBATCH --time=10:00
#SBATCH --nodes=1
#SBATCH --gres=gpu:1
#SBATCH --output=output.log
export PATH=$PATH:$HOME/ollama/bin # or: module load ollama
# Create a temp directory
export TMPDIR=$SCRATCH/ollama_tmp
mkdir -p $TMPDIR
# Choose a random available port for the ollama service
while true; do
    OLLAMA_PORT=`shuf -i 10000-20000 -n 1`
    ss -lpth | grep -q ":$OLLAMA_PORT " || break
done
export OLLAMA_HOST=127.0.0.1:${OLLAMA_PORT}
# Start Ollama service
ollama serve &> serve_ollama_${SLURM_JOBID}.log &
# Wait until Ollama service has been started
sleep 20
# Run Ollama using llama3 model
ollama run llama3.1:8b "How do you schedule a job with slurm?"
# Terminate Ollama service
killall ollama
```

- Loads latest version of ollama module.
- Sets Ollama to use a random available port
- Starts the ollama service in the background and redirects both standard output and standard error to a log file named **serve_ollama_{SLURM_JOBID}.log**.
- Executes the ollama run command using the model llama3.1:8b. If the model is not already installed in the user's account, the system will first execute ollama pull to download the model. This may add some time to the process
- You may submit multiple prompts
- The job exits as soon as it is completed

SBATCH options

Examples of SBATCH options:

1. `--job-name`: defines the name of the job to be submitted
2. `--partition`: specifies the queue in which the job will be submitted
3. `--time`: determines the maximum time we need to complete the job
4. `--output`: specifies the path where the output channel (STDOUT) of our batch job will appear

```
### Batch Script
```

```
#!/bin/bash
```

```
#SBATCH option1=value1
```

```
#SBATCH option2=value2
```

```
#!/bin/bash
#SBATCH --job-name=Ollama-batch
#SBATCH --partition=ampere
#SBATCH --time=10:00
#SBATCH --nodes=1
#SBATCH --gres=gpu:1
#SBATCH --output=output.log
```

Batch job - output

Submit a job

Tail the .log file of Ollama service

```
[cmosch@aristotle4 batchJob]$ sbatch batchJob_BashScript.sh
Submitted batch job 1794212
[cmosch@aristotle4 batchJob]$ tail -f serve_ollama_1794212.log
time=2024-04-16T13:33:31.391+03:00 level=INFO source=dyn_ext_server.go:159 msg="Starting llama main loop"
{"function":"update_slots","level":"INFO","line":1574,"msg":"all slots are idle and system prompt is empty"}
{"function":"launch_slot_with_data","level":"INFO","line":826,"msg":"slot is processing task","slot_id":1}
{"function":"update_slots","ga_i":0,"level":"INFO","line":1805,"msg":"slot progression","n_past":0,"n_processed":1713263611}
13263615,"truncated":false}
GIN] 2024/04/16 - 13:33:35 | 200 | 6.925133904s | 127.0.0.1 | POST "/api/generate"
C
```

Tail the .log file of the submitted job

POST command on Ollama API

```
[cmosch@aristotle4 batchJob]$ tail -f output.log
Once you have written your `sbatch` file, you can submit it to the cluster using the following command:
```
sbatch your_job_file.txt
```
This will submit the job to the cluster and it will be executed on the specified number of nodes with the specified CPUs per node, and any errors will be saved in the error log file.
```

Ollama Python library

- The Ollama Python library is a convenient toolkit that allows you to directly utilize the available Large Language Models in Ollama within your Python scripts. It can be installed with pip:

\$ pip install ollama

Note: The Ollama service needs to be running...

```
#!/usr/bin/env python3

import ollama
import os

ollama_host = os.getenv('OLLAMA_HOST')

client = ollama.Client(host=ollama_host)
response = client.chat(model='llama3', messages=[
    {
        'role': 'user',
        'content': 'What popular operating system, launched in 1991, \
                    also has its own mascot, Tux the penguin?'
    }
])

print(response['message']['content'])
```

A small example on how to run Llama3 from the Ollama Python library

Batch job with Python script - Python environment

How to create a virtual environment including ollama

```
[cmosch@aristotle4 batchJob]$ module load gcc/12.2.0 python/3.10.10
[cmosch@aristotle4 batchJob]$ python -m venv myenv
[cmosch@aristotle4 batchJob]$ source myenv/bin/activate
(myenv) [cmosch@aristotle4 batchJob]$ pip install --upgrade pip
Requirement already satisfied: pip in ./myenv/lib/python3.10/site-packages
Collecting pip
  Using cached pip-24.0-py3-none-any.whl (2.1 MB)
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 22.3.1
    Uninstalling pip-22.3.1:
      Successfully uninstalled pip-22.3.1
  Successfully installed pip-24.0
(myenv) [cmosch@aristotle4 batchJob]$ pip install ollama pandas jupyter
Collecting ollama
  Using cached ollama-0.1.8-py3-none-any.whl.metadata (3.8 kB)
Collecting pandas
```

- Simplest way → use Python's built-in venv module
- Create and activate the environment myenv
- Useful to upgrade Python's standard package manager (pip) to the latest available version before proceeding with the installation of additional packages
- Install via pip all the desired libraries

More information on

<https://hpc.it.auth.gr/languages/python/>

Batch job with Python script - batch script

Python script and ollama python library

- Starts the ollama service and pipes its output to both the console and a log file named **serve_ollama_\${SLURM_JOBID}.log**
- Activates the Python virtual environment where ollama and its dependencies are installed
- Executes a Python script named *test_ollama.py* with unbuffered output and redirects its output to a file named *python_script_output.txt*

```
#!/bin/bash
#SBATCH --job-name=Ollama-batch
#SBATCH --partition=ampere
#SBATCH --time=10:00
#SBATCH --nodes=1
#SBATCH --gres=gpu:1
#SBATCH --output=output.log
export PATH=$PATH:$HOME/ollama/bin
# Create a temp directory
export TMPDIR=$SCRATCH/ollama_tmp
mkdir -p $TMPDIR
# Choose a random available port for the ollama service
while true; do
  OLLAMA_PORT=`shuf -i 10000-20000 -n 1`
  ss -ltna | grep -q ":$OLLAMA_PORT " || break
done
export OLLAMA_HOST=127.0.0.1:${OLLAMA_PORT}
# Start Ollama service
ollama serve &> serve_ollama_${SLURM_JOBID}.log &
# Wait until Ollama service has been started
sleep 20
# Activate the virtual environment
source ollama_env/bin/activate
# Run the python script
python -u test_ollama.py > python_script_output.txt
# Terminate Ollama service
killall ollama
```

How to run LLMs on Aristotelis HPC infrastructure

- Available Partitions
- Ollama Installation
- OnDemand
- Batch jobs
- **Interactive session**
- REST API
- Jupyter notebook
- Clients and UIs
- Potential Issues

Interactive Session - batch script

You can run the Ollama service on the HPC cluster and the client on a login node!

```
#!/bin/bash
#SBATCH --job-name=Ollama-service
#SBATCH --partition=ampere
#SBATCH --time=10:00
#SBATCH --nodes=1
#SBATCH --gres=gpu:1
#SBATCH --output=output.log
export PATH=$PATH:$HOME/ollama/bin
# Create a temp directory
export TMPDIR=$SCRATCH/ollama_tmp
mkdir -p $TMPDIR
# Choose a random available port for the ollama service
while true; do
  OLLAMA_PORT=`shuf -i 10000-20000 -n 1`
  ss -ltn | grep -q ":$OLLAMA_PORT " || break
done
# Retrieve the public IP address of the host
IP_ADDR=$(curl ip.me)
# Set the OLLAMA_HOST variable
export OLLAMA_HOST=${IP_ADDR}:${OLLAMA_PORT}
# Send a push notification to my phone (see https://ntfy.sh for more)
curl -d "${SLURM_JOBID}@${OLLAMA_HOST}:${OLLAMA_PORT}." ntfy.sh/ollama123
# Start Ollama service
ollama serve &> serve_ollama_${SLURM_JOBID}.log
```

- Retrieves the public IP address of the host using curl from the ip.me service and sets it as the OLLAMA_HOST environment variable
- Starts the ollama service and pipes its output to both the console and a log file named **serve_ollama_\${SLURM_JOBID}.log**
- You don't run **ollama run** in the batch script in this case

Interactive Session

- Submit the job:

```
$ sbatch ollama_service.sh
Submitted batch job 1796446
$ tail -f serve_ollama_1796446.log
time=2024-04-24T08:43:49.320+03:00 level=INFO source=images.go:817 msg="total blobs: 44"
time=2024-04-24T08:43:49.356+03:00 level=INFO source=images.go:824 msg="total unused blobs removed: 0"
time=2024-04-24T08:43:49.360+03:00 level=INFO source=routes.go:1143 msg="Listening on 155.207.96.50:12906 (version 0.1.32)"
```

- Connect to it from the login node:

```
$ export OLLAMA_HOST=155.207.96.50:12906
$ ollama run llama3
>>> Send a message (/? for help)
```

Make sure you connect to the right IP and port from the login node!

How to run LLMs on Aristotelis HPC infrastructure

- Available Partitions
- Ollama Installation
- OnDemand
- Batch jobs
- Interactive session
- **REST API**
- Jupyter notebook
- Clients and UIs
- Potential Issues

REST API

- Remember that **ollama** runs a service component. What you get with **ollama run** is just a client. The simplest client is a just a REST API
- Generate a response:

```
curl http://155.207.96.50:11434/api/generate -d '{ "model": "llama3", \
  "prompt": "Why is the sky blue?" }'
```

155.207.96.50 is the IP of the HPC node that runs the ollama service in this example and 11434 is the port. Substitute them for the ones you're using.

- Chat with a model:

```
curl http://155.207.96.50:11434/api/chat -d '{ "model": "llama3", \
  "messages": [{"role": "user", "content": "Why is the sky blue?"}]}'
```

REST API - example output

```
{ "model": "llama3", "created_at": "2024-04-20T20:24:08.725750389Z", "response": "What", "done": false }
{ "model": "llama3", "created_at": "2024-04-20T20:24:08.882499464Z", "response": " a", "done": false }
{ "model": "llama3", "created_at": "2024-04-20T20:24:09.036335811Z", "response": " great", "done": false }
{ "model": "llama3", "created_at": "2024-04-20T20:24:09.191317995Z", "response": " question", "done": false }
{ "model": "llama3", "created_at": "2024-04-20T20:24:09.354884382Z", "response": "!\\n\\n", "done": false }
{ "model": "llama3", "created_at": "2024-04-20T20:24:09.518166884Z", "response": "The", "done": false }
{ "model": "llama3", "created_at": "2024-04-20T20:24:09.673687543Z", "response": " sky", "done": false }
{ "model": "llama3", "created_at": "2024-04-20T20:24:09.827425264Z", "response": " appears", "done": false }
{ "model": "llama3", "created_at": "2024-04-20T20:24:09.983749935Z", "response": " blue", "done": false }
{ "model": "llama3", "created_at": "2024-04-20T20:24:10.137785563Z", "response": " because", "done": false }
{ "model": "llama3", "created_at": "2024-04-20T20:24:10.293128093Z", "response": " of", "done": false }
{ "model": "llama3", "created_at": "2024-04-20T20:24:10.449808154Z", "response": " a", "done": false }
...
```

* REST API can be accessed from login nodes. For **ampere** and **ondemand** it can also be accessed from all AUTH network for ports 11434-11443.

How to run LLMs on Aristotelis HPC infrastructure

- Available Partitions
- Ollama Installation
- OnDemand
- Batch jobs
- Interactive session
- REST API
- **Jupyter notebook**
- Clients and UIs
- Potential Issues

Jupyter Notebook (1/3)

- Start ollama service with the same batch script as in the Interactive Session. Be sure to note the host ip!
- Start a Jupyter Server on the cluster (Through the menu on <https://hpc.auth.gr> choose Interactive Apps -> Jupyter)

Jupyter Server

This app will launch a [Jupyter](#) server on [Aristotelis cluster](#).

Please, visit our [Jupyter documentation page](#) for information on installing and loading python packages.

Number of hours

Number of cores

Max: 8 CPU cores (memory per core: 3GB)

I would like to receive an email when the session starts

Launch

* The Jupyter Server session data for this session can be accessed under the [data root directory](#).

Jupyter Notebook (2/3)

- Read the instructions at <https://hpc.it.auth.gr/applications/jupyter/> on how to setup a Python virtual environment
- Make sure you install the ollama library

```
mkdir envs
cd envs
python -m venv ollama_env
source ollama_env/bin/activate
pip install --upgrade pip
pip install ipykernel
python -m ipykernel install --user --name my-custom-env --display "Ollama env"
pip install ollama
```


Jupyter Notebook (3/3)

- Create a new notebook using the new ollama environment

Import ollama library

```
In [1]: from ollama import Client
```

Connect to the host of the ollama service

```
In [2]: host_ip = 'http://155.207.96.50:11434'  
client = Client(host=host_ip)
```

*Ollama service node
IP and port*

Chat with a model

```
In [4]: response = client.chat(model='llama3', messages=[  
    { 'role': 'user', 'content': 'How does distributed computing differ from cloud computing?' }, ] )  
print(response['message']['content'])
```

Distributed computing and cloud computing are two distinct concepts that often get confused with each other. Here's how they differ:

****Distributed Computing:****

Distributed computing refers to a system where multiple computers or nodes work together as a single, cohesive unit to accomplish a task or solve a problem. These nodes can be located anywhere in the world and may not necessarily be physically connected. Each node can have its own processor, memory, and storage, and they communicate with each other through networks.

In distributed computing, the focus is on breaking down complex tasks into smaller subtasks that can be executed concurrently.

How to run LLMs on Aristotelis HPC infrastructure

- Available Partitions
- OnDemand
- Batch jobs
- Interactive session
- Jupyter notebook
- REST API
- **Clients and UIs**
- Potential Issues

Clients and UIs

You can build other clients on top of the REST API. There are several different ones available

- Web UI
- Terminal
- Editor
- Mobile
- Plugins
- Libraries

There is a long list at <https://github.com/ollama/ollama>

Clients and UIs - batch script

You'll also need to set the `OLLAMA_ORIGINS="*"` variable in your batch script to allow access from web browsers. It's a [CORS](#) issue.

Note the smaller range of ports.

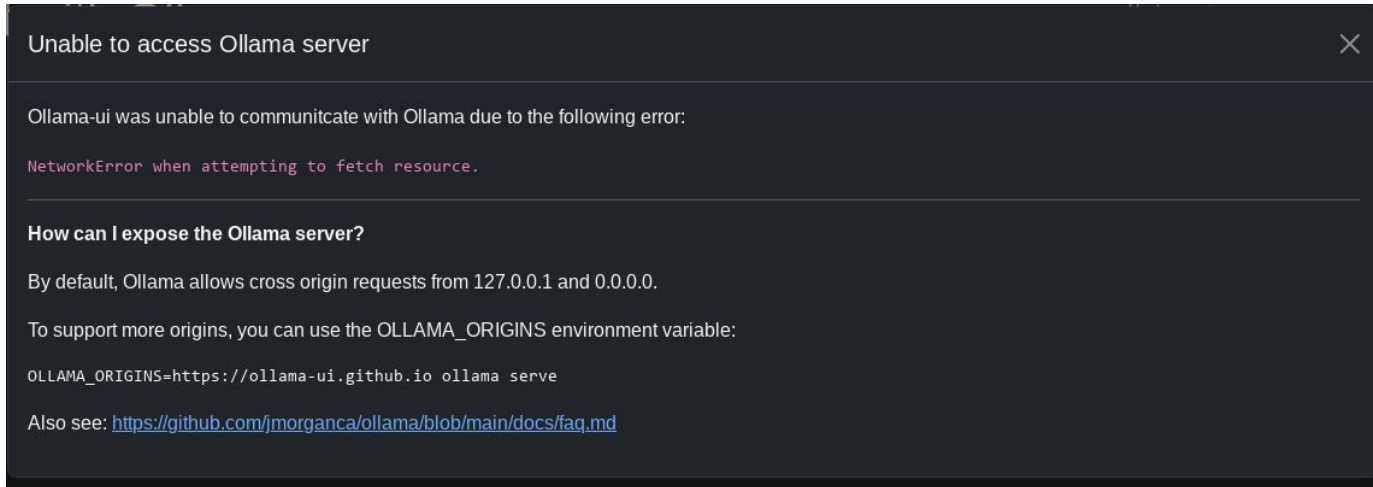
```
#!/bin/bash
#SBATCH --job-name=Ollama-service
#SBATCH --partition=ampere
#SBATCH --time=10:00
#SBATCH --nodes=1
#SBATCH --gres=gpu:1
#SBATCH --output=output.log
export PATH=$PATH:$HOME/ollama/bin
# Create a temp directory
export TMPDIR=$SCRATCH/ollama_tmp
mkdir -p $TMPDIR
# Choose a random available port for the ollama service
while true; do
    OLLAMA_PORT="`shuf -i 11434-11443 -n 1`"
    ss -ltn | grep -q ":$OLLAMA_PORT " || break
done
# Retrieve the public IP address of the host
IP_ADDR=$(curl ip.me)
# Set the OLLAMA_HOST variable
export OLLAMA_HOST=${IP_ADDR}:${OLLAMA_PORT}
# Allow browsers to connect to the service
export OLLAMA_ORIGINS="*"
# Send a push notification to my phone (see https://ntfy.sh for more)
curl -d "${SLURM_JOBID}@${OLLAMA_HOST}:${OLLAMA_PORT}." ntfy.sh/ollama123
# Start Ollama service
ollama serve &> serve_ollama_${SLURM_JOBID}.log
```

Clients and UIs - Webapp (1/6)

A simple UI is available as a webapp at <https://ollama-ui.github.io/ollama-ui/>

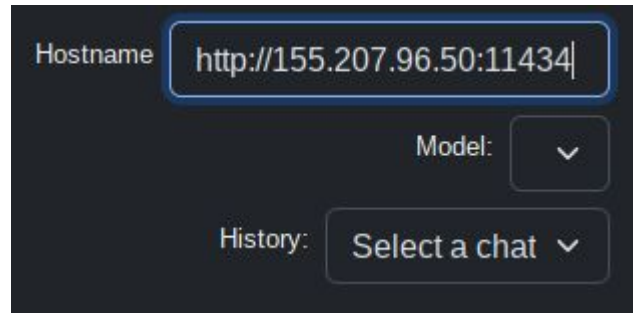
- Works with Firefox and Chrome

You will get this error message when you load the webapp. That's OK, close the message...



Clients and UIs - Webapp (2/6)

You need to enter the ollama service IP address along with the port in the format *http://ip_address:port*

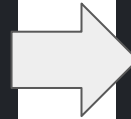


A screenshot of a dark-themed web application interface. It features three configuration fields: a text input for 'Hostname' containing 'http://155.207.96.50:11434', a dropdown menu for 'Model' with a downward arrow, and another dropdown menu for 'History' with the text 'Select a chat' and a downward arrow.

Clients and UIs - Webapp (3/6)

Firefox

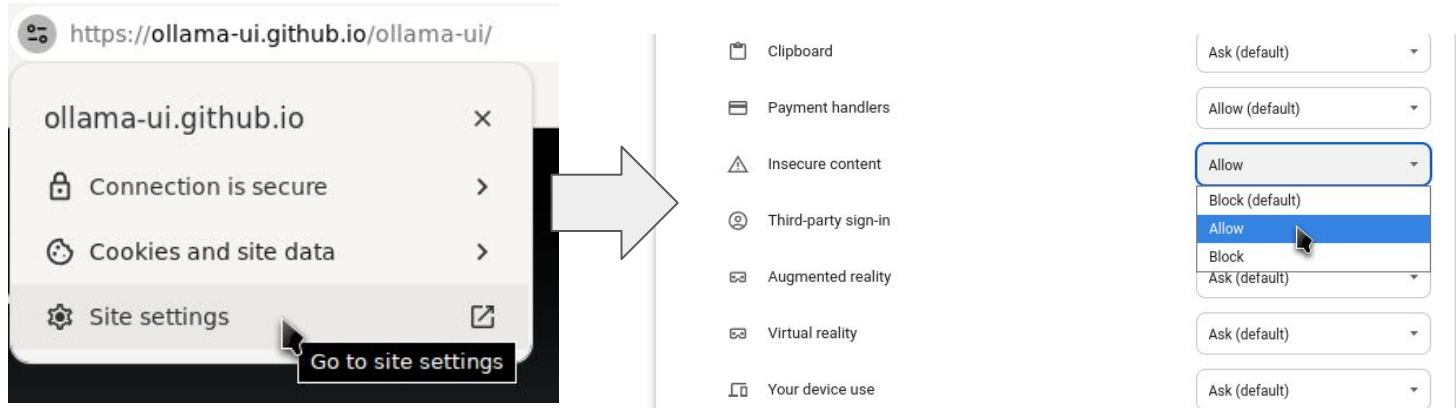
Click on the lock icon on the address bar and select to *Disable protection for now* (it's a “Mixed content” issue, loading http content from an https page)



Clients and UIs - Webapp (4/6)

Chrome

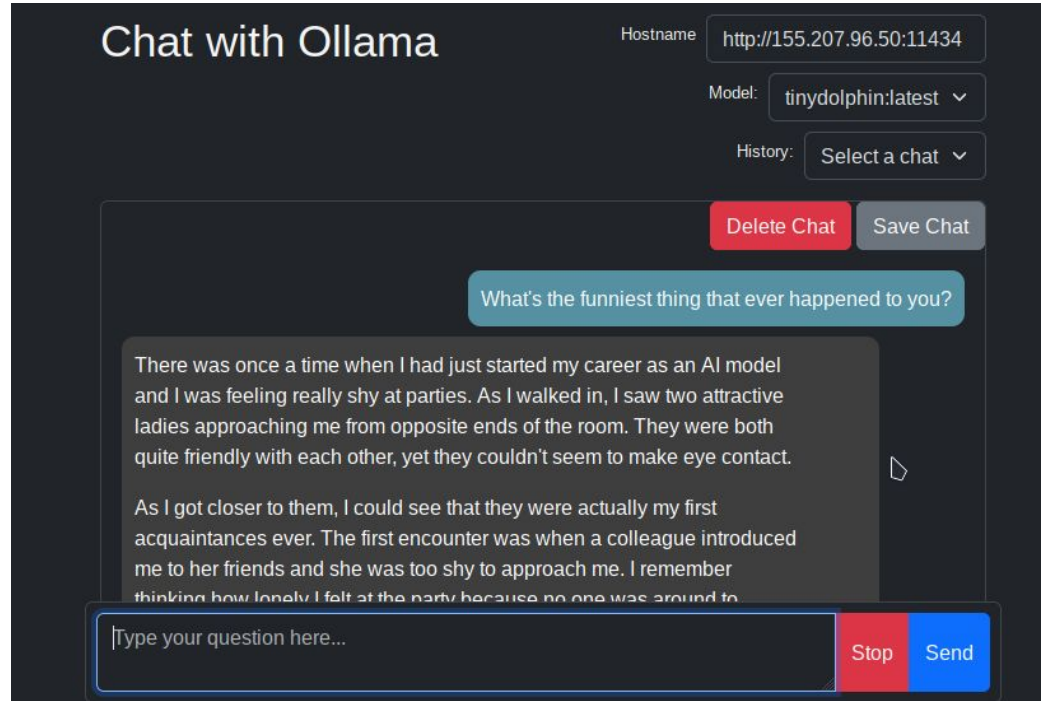
Click on the lock icon on the address bar and go to *Site Settings*. Then set *Insecure content* to *Allow*.



Clients and UIs - Webapp (5/6)

And then it works!

- The available models will be shown in the *Model* dropdown and you can select the one you'd like to use.
- You can save any conversations you'd like to keep



Clients and UIs - Webapp (6/6)

There are better UIs. Recommended:

- OpenWebUI: <https://github.com/open-webui/open-webui>
- BionicGPT: <https://github.com/bionic-gpt/bionic-gpt>
- AnythingLLM: <https://github.com/Mintplex-Labs/anything-llm>

But they require some (minimal?) effort in installation.

How to run LLMs on Aristotelis HPC infrastructure

- Available Partitions
- OnDemand
- Batch jobs
- Interactive session
- Jupyter notebook
- REST API
- Clients and UIs
- **Potential Issues**

Potential Issues

- Not enough HD space for big models
 - Default `$HOME` for users is 20GB
 - Large models can't fit
- Possible solutions:
 - Request more space for your account: <https://hpc.it.auth.gr/home-directories/>
 - Store the `~/.ollama` directory under `$SCRATCH` and symlink it from there
 - Files get deleted from `$SCRATCH` after 30 days

```
$ mkdir -p $SCRATCH/ollama-models
$ ln -s $SCRATCH/ollama-models ~/.ollama
```

Performance Comparisons

- Model Performance on Aristotelis

Model performance on Aristotelis

	ampere	gpu	ondemand	a4000	rome	batch
Model	Tokens/s (Layers in GPU)	Tokens/s (Layers in GPU)	Tokens/s (Layers in GPU)	Tokens/s (Layers in GPU)	Tokens/s	Tokens/s
llama3.2:1b	189 (17/17)	90 (17/17)	136 (17/17)	128 (17/17)	8.8	7.6
llama3.2:3b	153 (29/29)	53 (29/29)	86 (29/29)	90 (29/29)	7.1	4.4
qwen2.5:7b	110 (29/29)	31 (29/29)	<0.1 (27/29)	59 (29/29)	3.9	3.2
llama3.1:8b	110 (33/33)	30 (33/33)	<0.1 (28/33)	60 (33/33)	3.2	2.9
qwen2.5:14b	40 (49/49)	15 (49/49)	<0.1 (21/49)	31 (49/49)	2.3	?
qwen2.5:32b	28 (65/65)	<0.1 (35/65)	?	<0.1 (49/65)	?	?
llama3.3:70b*	24.5 (81/81)	?	?	?	?	?

Model performance on Aristotelis - GPU vs CPU

- Running on GPUs is considerably faster!
- 7b and 8b models can run adequately on CPUs. If your local PC is not capable of running them, they can provide an alternative
- On CPUs, you should make sure to run the process **in a single socket**. Otherwise performance will suffer.

```
#!/bin/bash
#SBATCH --job-name=Ollama-cpu-serve
#SBATCH --partition=rome
#SBATCH --time=20:00
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=64
#SBATCH --extra-node-info=1:64
#SBATCH --output=output-rome.log
```

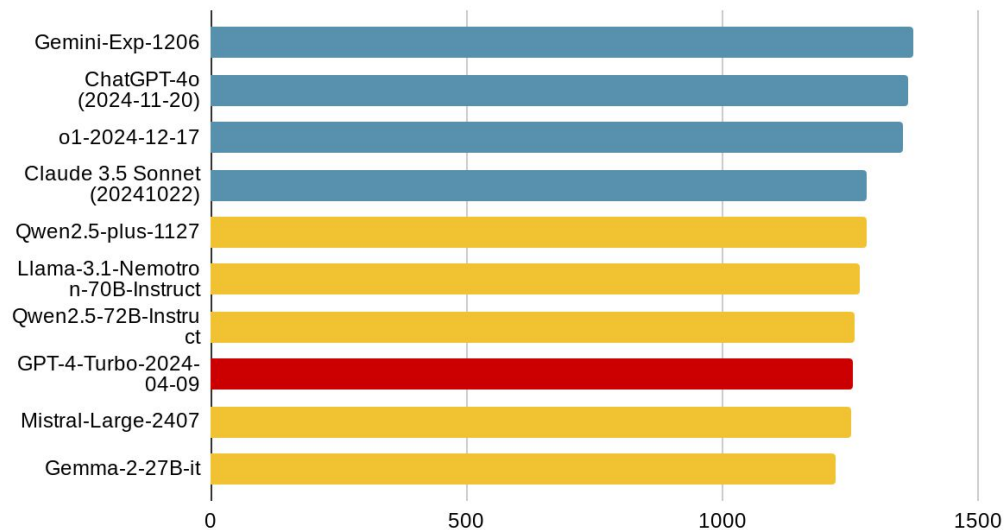
Comparison of different models: from small to large

- LLM Leaderboard
- Model Overview

LLM Leaderboard (1/2)

LMSYS Chatbot Arena

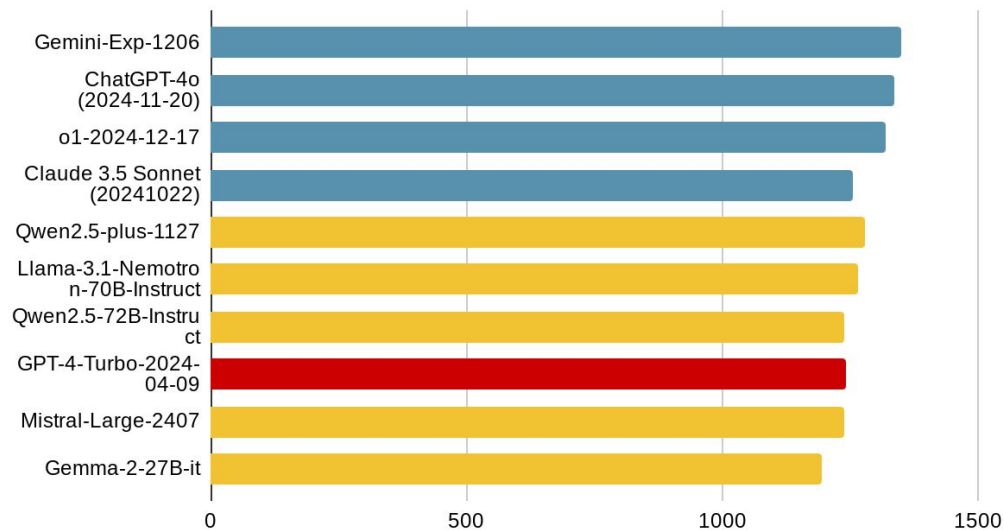
Overall



LLM Leaderboard (2/2)

LMSYS Chatbot Arena

English



Model Overview

- Small models (1-3b):
 - Designed to be efficient and compact
 - Useful when accuracy is not the primary concern
 - Can be easily fine-tuned to specific uses
 - Text classification
 - Sentiment analysis
 - Basic chatbots
 - Simple definitions
 - Grammar/syntax corrections
 - Can be used even in underpowered devices
- Medium models (~7-13b)
 - Improved task performance
 - Enhanced contextual understanding
 - Increased ability to generalize
 - Efficient on average modern PCs
 - Very fast on high-end hardware
- Larger Models ($\geq 70b$):
 - State-of-the-art
 - Deep understanding of complex patterns and relationships
 - Capacity for abstraction
 - May rival top proprietary LLMs
 - Need high-end hardware to operate efficiently
- Size matters, but it's not everything
- Things are progressing at an incredible pace

Thank you!

Questions?