Large Language Models

- How to run LLM models
- Performance of LLM backends and models in Curnagl

How to run LLM models

This tutorial shows, how to run LLM on UNIL clusters

Available models

You are free to download and use any LLM model you like in your `/work` space, but the process can sometimes be a bit tricky. In addition, several users may want to use the same models.

To simplify this, we now provide a selection of LLM models for the whole community. These models are obtained directly from the Hugging Face Hub.

Location: `/reference/LLM/`

Naming convention

Folder names follow Hugging Face terminology. For example, the folder `meta-llama/Llama-3.1-8B-Instruct` corresponds to the model available at: https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct

If the LLM you need is not available, you can request its installation by sending an email to `helpdesk@unil.ch` with the subject line: DCSR Request LLM installation.

Simple test

set up

For this simple test, we are going to use `transformers` library from hugging face. So you should type the following commands to setup a proper python environment:

```
module load python
python -m venv venv
source venv/bin/activate
pip install transformers accelerate torch
```

If you plan to use a instruct model, you will need a chat template file which you can download from https://github.com/chujiezheng/chat_templates . For this example, we are going to use the llama template

```
wget
https://raw.githubusercontent.com/chujiezheng/chat_templates/refs/heads/main/chat_templates/ll
ama-3-instruct.jinja
```

Then, you should create the following python file:

```
from transformers import AutoModelForCausalLM
from transformers import AutoTokenizer

model_hf='/reference/LLM/meta-llama/Llama-3.1-8B-Instruct/'

model = AutoModelForCausalLM.from_pretrained(model_hf, device_map="auto")
tokenizer = AutoTokenizer.from_pretrained(model_hf)

with open('llama-3-instruct.jinja', "r") as f:
    chat_template = f.read()

tokenizer.chat_template = chat_template

with open('prompt.txt') as f:
    prompt=f.read()

prompts = [
    [{'role': 'user', 'content': prompt}]
]

model_inputs = tokenizer.apply_chat_template(
    prompts,
    return_tensors="pt",
    tokenize=True,
    add_generation_prompt=True #This is for adding prompt, useful in chat mode
).to("cuda")

generated_ids = model.generate(
    model_inputs,
    max_new_tokens=400,
```

```
)

for i, answer in enumerate(tokenizer.batch_decode(generated_ids, skip_special_tokens=True)):
    print(answer)
```

This python code reads a prompt from a text file called prompt.tx and uses the model 8B de LLama to perform the inference.

To turn it in the cluster, we can use the following job script:

```
#!/bin/bash

#SBATCH -p gpu
#SBATCH --mem 20G
#SBATCH --gres gpu:1
#SBATCH -c 2

source venv/bin/activate
python run_inference.py
```

You should run the previous command sbatch:

```
sbatch job.sh
```

The result of the inference will be written in the SLURM file `slurm-xxxx.out`

Using VLLM

If you need to run big models, you can use VLLM library which uses less GPU memory. To install it:

```
pip install vllm
```

Then, you can use it with the following simple code:

```
import os
import time
from vllm import LLM, SamplingParams

num_gpus =
len(os.environ['CUDA_VISIBLE_DEVICES'].split(","))
```

```
model_hf='/reference/LLM/meta-llama/Llama-3.1-8B-Instruct/'

with open('llama-3-instruct.jinja', "r") as f:
    chat_template = f.read()

with open('prompt.txt') as f:
    prompt=f.read()

prompts = [
    [{'role': 'user', 'content': prompt}]
]

model = LLM(model=model_hf,tensor_parallel_size=num_gpus)

sampling = SamplingParams(
    n=1,
    temperature=0,
    max_tokens=400,
    skip_special_tokens=True,
    stop=["<|eot_id|>"]
)

output = model.chat(
    prompts, sampling, chat_template=chat_template,
)



results = []
for i, out in enumerate(output):
    answer = out.outputs[0].text
    print(answer)
```

If you need to use several GPUs, do not forget to put `#SBATCH --gres gpu:2` in your job description

Performance of LLM backends and models in Curnagl

Introduction

This page shows performance of Llama and mistral models on Curnagl hardware. We have measured the token throughput which should help you to have an idea of what is possible using Curnagl resources. Training time and inference time for different task could be estimated using these results.

Models and backends tested

Tested Models

Llama3

- Official access to Meta Llama3 models: Meta Llama3 models on Hugging Face
- Meta-Llama-3.1-8B-Instruct
- Meta-Llama-3.1-70B-Instruct

Mistral

- Official access to Mistral models: Mistral models on MistralAI website
- Access to Mistral models on Hugging Face: Mistral models on Hugging Face
- mistral-7B-Instruct-v0.3
- Mixtral-8x7B-v0.1-Instruct

Tested Backends

- **vLLM**

vLLM backend provides efficient memory usage and fast token sampling. This backend is ideal for testing Llama3 and Mistral models in environments that require high-speed responses and low latency.

- **llama.cpp**

llama.cpp was primarily used for llama but it can be applied to other LLM models. This optimized backend provides efficient inference on GPUs.

- **Transformers**

If not the most widely used LLM black box, it is one of them. Easy to use, the Hugging Face Transformers library supports a wide range of models and backends. One of its main advantages is its quick set up, which enables quick experimentation across architectures.

- **mistral-inference**

This is the official inference backend for Mistral. It is (supposed to be) optimized for Mistral's architecture, thus increasing the model performance. However, our benchmarks results do not demonstrate any specificities to Mistral model as llama.cpp seems to perform better.

Hardware description

Three different types of GPUs have been used to benchmark LLM models:

- A100 which are available on Curnagl, official documentation,
- GH200 which will be available soon on Curnagl, official documentation,
- L40 which will be available soon on Curnagl, official documentation and specifications.

Here are their specifications

Characteristics	A100	GH200	L40S
Number of nodes at UNIL	8	1	8
Memory per node (GB)	40	80	48

Characteristics	A100	GH200	L40S
Number of CPU per NUMA node	48	72	8
Memory bandwidth - up to (TB/s)	1.9	4	0.86
FP64 performance (teraFlops)	9.7	34	NA
TF64 performance (teraFlops)	19.5	67	NA
FP32 performance (teraFlops)	19.5	67	91.6
TF32 performance (teraFlops)	156	494	183
TF32 performance with sparsity (teraFlops)	312	494	366
FP16 performance (teraFlops)	312	990	362
INT8 performance (teraFlops)	624	1.9	733

Depending on the code you are running, one GPU may better suit your requirements and expectations.

Note: These architectures are not powerful enough to train Large Language Models.

Note: Our benchmarks aim to determine which GPU types should be provided to researchers. If you require new GPUs for your research, feel free to reach out to us through the Help Desk. In case, you and other researchers agree on the same GPU request, we will do our best to provide new resources that meet your needs.

Inference latency results

This chat dataset from GPT3 has been used to benchmark models.

In order to guarantee reproduciblity of resultst and be able to perform a comparison between different bechmarks we set the following parameters:

- The maximum number of tokens to generate, is set to `400`
- The temperature, which controls the output randomness, is set to `0`

- The context size, which is the number of tokens the model can process within a single input, is set to `default`. This means the maximum context size of the model (e.g 131072 for Llama3.1)
- Use of GPU exclusively
- All models are loaded in F16 (no quantization)

Mistral models

mistral-7B-Instruct-v0.3

Backend results (Token/seconds)	A100	GH200	L40
vllm	74.1	-	-
llama.cpp	53.8	138.4	42.8
Transformers	30	41.3	21.6
mistral-inference	23.4	-	25

Mixtral-8x7B-v0.1-Instruct

Backend results (Token/seconds)	A100	GH200	L40
llama.cpp	NA	NA	23.4
Transformers	NA	NA	8.5

Llama models

8B Instruct

Backend results (Token/seconds)	A100	GH200	L40
llama.cpp	62.645	100.845	43.387
Transformers	31.650	43.321	21.062
vllm	44.686	119.59	45.176

70B Instruct

Backend results (Token/seconds)	L40

llama.cpp	5.029
Transformers	2.372
vllm	30.945

Conclusions

- Mixtral 8x7B and Llama 70B Instruct are composed of several billions of parameters. Therefore the resulting memory consumption for inference can only be supported by multiple GPUs using the same machine or by using a combination of VRAM and RAM host memory. This of course will degrade the performance because we need to transfert data between two types of memory which could be slow. GH200 has a large bus memory which offers a good performance on this types of cases.
- The use of distributed setup adds a lot of latency.
- Transformers backend offers a good trade-off between learning curve and performance.
- Banckends offer the possibility to configure a context size. The parameter has no impact on peformance (token throughput) but it is correlated to the amount of VRAM consumed. Therefore, if you want to optimize memory consumption you should set the context size to an appropiate value.
- GH200 offers the best inference speed but it could be difficult to set up and install libraries on.
- The results shown here were obtained without any optimization. There are optimization than can be applied like quantization and flash attention.