# **vLLM Benchmarking Procedure - Xeons**

The procedure is broadly divided into 4 steps:

- Getting Server Details   
- Making sure the BIOS is correctly setup   
- Configuring docker   
- Building vLLM image for Xeons   
- Run the benchmark setup with **Load Balancing**  
- Run the benchmark script with **Tensor Parallelism**

## **Server Details**

To get the server hardware details, just run the command: **lscpu** Here is a sample output:

Architecture: x86\_64  
 CPU op-mode(s): 32-bit, 64-bit  
 Address sizes: 52 bits physical, 57 bits virtual  
 Byte Order: Little Endian  
CPU(s): 288  
 On-line CPU(s) list: 0-287  
Vendor ID: GenuineIntel  
 Model name: GENUINE INTEL(R) XEON(R)  
 CPU family: 6  
 Model: 173  
 Thread(s) per core: 2  
 Core(s) per socket: 72  
 Socket(s): 2  
 Stepping: 1  
 CPU(s) scaling MHz: 21%  
 CPU max MHz: 3900.0000  
 CPU min MHz: 800.0000  
 BogoMIPS: 5600.00  
 Flags: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush amx\_bf16 avx512\_fp16 amx\_t ile amx\_int8 flush\_l1d arch\_capabilities  
Virtualization features:  
 Virtualization: VT-x  
Caches (sum of all):  
 L1d: 6.8 MiB (144 instances)  
 L1i: 9 MiB (144 instances)  
 L2: 288 MiB (144 instances)  
 L3: 864 MiB (2 instances)  
NUMA:  
 NUMA node(s): 6  
 NUMA node0 CPU(s): 0-23,144-167  
 NUMA node1 CPU(s): 24-47,168-191  
 NUMA node2 CPU(s): 48-71,192-215  
 NUMA node3 CPU(s): 72-95,216-239  
 NUMA node4 CPU(s): 96-119,240-263  
 NUMA node5 CPU(s): 120-143,264-287  
Vulnerabilities:  
 Gather data sampling: Not affected  
 Itlb multihit: Not affected  
 L1tf: Not affected  
 Mds: Not affected  
 Meltdown: Not affected  
 Mmio stale data: Not affected  
 Reg file data sampling: Not affected  
 Retbleed: Not affected  
 Spec rstack overflow: Not affected  
 Spec store bypass: Mitigation; Speculative Store Bypass disabled via prctl  
 Spectre v1: Mitigation; usercopy/swapgs barriers and \_\_user pointer sanitization  
 Spectre v2: Mitigation; Enhanced / Automatic IBRS; IBPB conditional; RSB filling; PBRSB-eIBRS Not affected; BHI BHI\_DIS\_S  
 Srbds: Not affected  
 Tsx async abort: Not affected

The important information here that should be noted for our procedure is the NUMA Node configuration:

NUMA:  
 NUMA node(s): 6  
 NUMA node0 CPU(s): 0-23,144-167  
 NUMA node1 CPU(s): 24-47,168-191  
 NUMA node2 CPU(s): 48-71,192-215  
 NUMA node3 CPU(s): 72-95,216-239  
 NUMA node4 CPU(s): 96-119,240-263  
 NUMA node5 CPU(s): 120-143,264-287

To get the server software details, run hostnamectl

Static hostname: elfleet048  
 Icon name: computer-server  
 Chassis: server 🖳  
 Machine ID: 5ced64b850fc4653a767cf82cebca465  
 Boot ID: a0f2cc1786aa4620ba3d96f64dc9c70c  
Operating System: Ubuntu 24.04.2 LTS  
 Kernel: Linux 6.8.0-55-generic  
 Architecture: x86-64  
 Hardware Vendor: Intel Corporation  
 Hardware Model: AvenueCity  
Firmware Version: BHSDCRB1.86B.3544.P02.2409040029  
 Firmware Date: Wed 2024-09-04  
 Firmware Age: 6month 3d

Here you can see the Operating System and the Kernel

## **BIOS Settings**

To be Completed.

## **Docker Setup**

The official instructions for Docker can be found on their website [here](https://docs.docker.com/engine/install/). You can select the Operating System we found above and follow from there. For Ubuntu, here are the instructions:

# Uninstall any docker libraries already present  
for pkg in docker.io docker-doc docker-compose docker-compose-v2 podman-docker containerd runc; do sudo apt-get remove $pkg; done  
  
```bash  
# Export proxies if any  
export http\_proxy=http://proxy.example:123/  
export https\_proxy=http://proxy.example:123/  
export no\_proxy=localhost,127.0.0.1,0.0.0.0  
  
# Add Docker's official GPG key:  
sudo apt-get update  
sudo apt-get install ca-certificates curl  
sudo install -m 0755 -d /etc/apt/keyrings  
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc  
sudo chmod a+r /etc/apt/keyrings/docker.asc  
  
# Add the repository to Apt sources:  
echo \  
 "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu \  
 $(. /etc/os-release && echo "${UBUNTU\_CODENAME:-$VERSION\_CODENAME}") stable" | \  
 sudo tee /etc/apt/sources.list.d/docker.list > /dev/null  
sudo apt-get update  
  
# Install packages  
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin  
  
# Verify  
sudo docker run hello-world  
  
# Add your user to docker group  
sudo usermod -aG docker $USER

Re-login to your server. You should be able to run docker without sudo.

## **Proxy Setup for Docker**

If you are behind a corporate network, you would need to configure docker to use proxies. These steps are tested on Ubuntu, but should work on other OS as well:

# Switch to root user  
sudo su

# Create systemd dir  
mkdir /etc/systemd/system/docker.service.d  
  
# Add proxies  
cat <<EOT >> /etc/systemd/system/docker.service.d/http-proxy.conf  
[Service]  
Environment="HTTP\_PROXY=http://proxy.example:123/"  
Environment="HTTPS\_PROXY=http://proxy.example:123/"  
Environment="NO\_PROXY=localhost,127.0.0.0"  
EOT  
  
# Reload daemon  
systemctl daemon-reload  
  
# Restart docker  
systemctl restart docker

## **Building vLLM image**

Clone the vLLM repository:

git clone https://github.com/vllm-project/vllm

If using a corporate proxy, export these variables:

export http\_proxy=http://proxy.example:123/  
export https\_proxy=http://proxy.example:123/  
export no\_proxy=localhost,127.0.0.1,0.0.0.0

Build the image

cd vllm  
docker build --build-arg https\_proxy=$https\_proxy --build-arg http\_proxy=$http\_proxy -f Dockerfile.cpu -t vllm-cpu-env .

## **Setup Python virtual environment**

Before executing any further commands, it’s good to create a fresh virtual environment   
for all your python dependencies. Execute these commands:  
  
# Create a new environment in your home directory  
python3 -m venv ~/vllm-bench  
  
If this command fails then you must install python3.x-venv using the command   
mentioned in the output. Once you do that, rerun the command above and follow next:  
  
# Activate your environment  
source ~/vllm-bench/bin/activate

## **Download Model**

It’s good to have the model downloaded before you start executing the benchmark.   
Once you activate your environment, follow these steps:  
  
# Install HF CLI  
pip install -U "huggingface\_hub[cli]"  
  
By default, the model will be downloaded in *~/.cache/huggingface/hub* directory. To use a different directory, you can export the HF\_CACHE variable:  
  
export HF\_CACHE=/path/to/model  
  
However, make sure to use the same path when generating the configuration files later. Now, to install the model:  
  
# Export your token  
export HF\_TOKEN=<token>  
  
# Download the model  
huggingface-cli download <model-name>

## **Installing LLMPerf**

## git clone https://github.com/ray-project/llmperf cd llmperf pip install -e . In case you have python version 3.12, this will result in an error. You can edit the *pyproject.toml* file in this directory and change the requirement, then rerun the *pip install -e .* command.

## **Run the benchmark script with Load Balancing**

Before following these steps, make sure we don’t have any containers running which can hinder performance.

In this procedure, we initiate multiple deployments of vLLM - one for each NUMA Node. For example, our NUMA Node configuration is:

NUMA:  
 NUMA node(s): 6  
 NUMA node0 CPU(s): 0-23,144-167  
 NUMA node1 CPU(s): 24-47,168-191  
 NUMA node2 CPU(s): 48-71,192-215  
 NUMA node3 CPU(s): 72-95,216-239  
 NUMA node4 CPU(s): 96-119,240-263  
 NUMA node5 CPU(s): 120-143,264-287  
  
Here we have hyper-threading enabled, due to which there are 2 sets of core ranges per NUMA Node. vLLM runs best when only one set of threads per NUMA Node are being   
used. To ensure this, only use the first part of each NUMA Node. In the above scenario, it will be: **0-23,24-47,48-71,72-95,96-119,120-143.** Thiswillbethestring we use for   
core-ranges in the script below.

We use a python script to generate a docker-compose.yml file which has configurations for multiple vLLM containers and a Nginx container. It also generates a nginx.conf file which contains the load balancing configuration across our vLLM containers.

#### Step 1 - Generate the configuration files

* Export your huggingface token (if required):  
  export HF\_TOKEN=<token>
* Download the Python script

wget https://raw.githubusercontent.com/akarX23/intel-scripts/refs/heads/master/LLMs/vllm-cpu-lb-bench/gen-deploy-files.py

* For a list of all options, run:  
  python3 gen-deploy-files.py --help
* Generate the configuration:

Use the core-ranges for each NUMA Node, as discussed above, specify a core-range for nginx, and specify a value of kv\_cache suitable for your RAM. The   
--kv\_cache value is in GBs, and this value multiplied by the number of   
core-ranges (comma separated) you have defined is the total space that will be occupied in your RAM. Keep at least 120GB of extra memory as an overhead.   
  
If you have used a custom directory for the models when downloading them above, make sure to point to them using the --hf\_cache option, otherwise the default *~/.cache/huggingface/hub* is used. Here is a sample run of the script:  
  
python3 gen-deploy-files.py --core\_ranges 0-23,24-47,48-71,72-95,96-119,120-143 --model meta-llama/Llama-3.1-8B-Instruct --docker\_image vllm-cpu-env --nginx\_port 8000 --nginx\_core 225-239 --kv\_cache 80 --hf\_cache /mnt/hf\_cache

This should generate the *docker-compose.yml* and *nginx.conf* files in the current directory.

#### Step 2 - Run the benchmark

* Install *numactl*sudo apt install -y numactl
* Download the benchmark script

wget <https://raw.githubusercontent.com/akarX23/intel-scripts/refs/heads/master/LLMs/vllm-cpu-lb-bench/bench.sh>  
  
chmod +x bench.sh

* Print the configurable options  
  ./bench.sh --help  
    
  # Output

Usage: ./bench.sh [OPTIONS]

Options:

--host Set the host (default: 0.0.0.0)

--port Set the port (default: 8000)

--deployment-files-root Set the deployment files root (default: /home/cefls\_user/intel-scripts/LLMs/vllm-cpu-lb-bench)

--dataset-name Set the dataset name (default: random)

--num-prompts Set the number of prompts (default: 1000)

--client-args Set additional client arguments

--model (Required) Set the model name

--concurrencies (Required) Set comma-separated concurrency values

--input-lengths (Required) Set comma-separated input length values

--output-lengths (Required) Set comma-separated output length values

--log-dir (Required) Set the log directory

--num-deployments (Required) Set the number of deployments

--cores-per-deployment (Required) Set the number of cores per deployment

--llmperf-root (Required) Set the LLMPerf root directory

-h, --help Show this help message and exit

To automate a long-running benchmark, the script accepts these arguments for the benchmark client:

--concurrencies <list> List of concurrency values (required, comma-separated)  
 --input-lengths <list> List of input token lengths (required, comma-separated)  
 --output-lengths <list> List of output token lengths (required, comma-separated)

The script will run every iteration possible of the list of numbers provided in these parameters, one after the other, and compile the results. For example, --concurrencies 1,2 --input-lengths 128,256 --output-lengths 512,1024 will run these combinations:

Concurrency,Input Length,Output Length   
1,128,512   
1,128,1024   
1,256,512   
1,256,512   
2,128,512   
2,128,1024   
2,256,512   
2,256,512

Lastly, the --log-dir parameter will accept a directory to save all logs to, which will be 3 files:

* vllm-server.out : Output of the vLLM Server command
* client.out : Output of the benchmark client
* results.csv : CSV formatted results for all the combinations specified

To record results, you need to specify --num-deployments and --cores-per-deployment.   
  
Also, --llmperf-root is the path to the LLMPerf directory.   
The --deployment-files-root is the directory where the generated *docker-compose.yml* and *nginx.conf* files live.  
  
Additionally, make sure to run the script on a separate set of threads, other than the ones allotted for the vLLM containers and NGINX using *numactl.*

A sample command to run a long-running benchmark:  
  
# HF Token for gated models  
export HF\_TOKEN=<hf-token>  
  
numactl -C 240-245 ./bench.sh --host localhost --port 8000 --deployment-files-root /home/cefls\_user/deploy --dataset-name random --num-prompts 300 --model meta-llama/Llama-3.1-8B-Instruct --concurrencies 128,256 --input-lengths 128,256 --output-lengths 512,1024 --log-dir run1 --num-deployments 6 --cores-per-deployment 24 --llmperf-root /home/cefls\_user/llmperf  
  
The results will be saved in the run1 directory as specified in the --log-dir parameter.

## **Run the benchmark script with Tensor Parallelism**

To run this script, we need first to start the vLLM container and login to its shell:

* Run the vLLM container

# Run the container  
docker run -d -v /home/$USER/.cache/huggingface/hub:/root/.cache/huggingface/hub -e http\_proxy=$http\_proxy -e HTTP\_PROXY=$http\_proxy -e https\_proxy=$https\_proxy -e HTTPS\_PROXY=$https\_proxy -e no\_proxy=$no\_proxy -e NO\_PROXY=$no\_proxy --name vllm --privileged --entrypoint sleep vllm-cpu-env infinity

* Login to its shell

docker exec -it vllm bash

Download the script

wget https://raw.githubusercontent.com/akarX23/intel-scripts/refs/heads/master/LLMs/vllm-master-bench.sh  
chmod +x vllm-master-bench.sh

Print the configurable options

./vllm-master-bench.sh --help  
  
# Output  
Usage: ./vllm-master-bench.sh [options]  
  
Options:  
 --host <host> vLLM server host (default: 0.0.0.0)  
 --port <port> vLLM server port (default: 8000)  
 --kv\_cache <GB> KV cache size in GB (default: 40)  
 --cpus\_bind <cpus> CPUs to bind (required)  
 --hf\_token <token> Huggingface token (default: $HF\_TOKEN)  
 --model <model> Model to host and benchmark (required)  
 --tp <integer> Tensor parallelism (default: 1)  
 --vllm-server-args <args> Extra arguments for vLLM server  
 --vllm-root <path> Root directory of vLLM (default: /home/akarx/intel-scripts/LLMs/vllm)  
 --dataset-name <name> Dataset for benchmarking (default: random)  
 --num-prompts <num> Number of prompts per request (default: 1000)  
 --concurrencies <list> List of concurrency values (required, comma-separated)  
 --input-lengths <list> List of input token lengths (required, comma-separated)  
 --output-lengths <list> List of output token lengths (required, comma-separated)  
 --client-args <args> Extra arguments for the benchmark client  
 --log-dir <path> Directory for logs (required)  
 --help Display this help message

When we run vLLM with tensor parallelism on CPUs, we treat each NUMA Node as a GPU Card and split the model amongst them. For example, our NUMA Node configuration is:

NUMA:  
 NUMA node(s): 6  
 NUMA node0 CPU(s): 0-23,144-167  
 NUMA node1 CPU(s): 24-47,168-191  
 NUMA node2 CPU(s): 48-71,192-215  
 NUMA node3 CPU(s): 72-95,216-239  
 NUMA node4 CPU(s): 96-119,240-263  
 NUMA node5 CPU(s): 120-143,264-287

Ideally, in our case, we should run vLLM with a maximum --tp value of 6. To do this, we use the --cpus\_bind parameter. Here we define a list of core ranges separated by “|” that will be used to split the model processing. **Note that the model memory is still loaded in the RAM, only the processing is split across the CPUS.**

Since we have hyper-threading enabled, each NUMA Node has 2 sets of CPU Threads. vLLM runs best when only one set of threads per NUMA Node is being used. In our server, if we want to run vLLM with --tp 6, the cpus\_bind will be like this: --cpus\_bind "0-23|24-47|48-71|72-95|96-119|120-143"

To automate a long-running benchmark, the script accepts these arguments for the benchmark client:

--concurrencies <list> List of concurrency values (required, comma-separated)  
 --input-lengths <list> List of input token lengths (required, comma-separated)  
 --output-lengths <list> List of output token lengths (required, comma-separated)

The script will run every iteration possible of the list of numbers provided in these parameters, one after the other, and compile the results. For example, --concurrencies 1,2 --input-lengths 128,256 --output-lengths 512,1024 will run these combinations:

Concurrency,Input Length,Output Length   
1,128,512   
1,128,1024   
1,256,512   
1,256,512   
2,128,512   
2,128,1024   
2,256,512   
2,256,512

Lastly, the --log-dir parameter will accept a directory to save all logs to, which will be 3 files: - vllm-server.out : Output of the vLLM Server command - client.out : Output of the benchmark client - results.csv : CSV formatted results for all the combinations specified

The other parameters are self-explanatory and can be used for experimentation to get different results.

A sample command you can run in the docker container for a long-running benchmark:

# If using a gated model  
export HF\_TOKEN=<your-hf-token>  
  
# Benchmark meta-llama/Llama-3.1-8B-Instruct  
./vllm-master-bench.sh --cpus\_bind "0-23|24-47|48-71|72-95|96-119|120-143" --tp 6 --model meta-llama/Llama-3.1-8B-Instruct --concurrencies 1,2,4,8,16,32,64 --input-lengths 128,256,512,1024 --output-lengths 128,256,512,1024 --log-dir run1

The results can be found in the run1 directory