The science behind the report:

Run your in-house AI chatbot on an AMD EPYC 9534 processor-powered Dell PowerEdge R6615 server

This document describes what we tested, how we tested, and what we found. To learn how these facts translate into real-world benefits, read the report Run your in-house AI chatbot on an AMD EPYC 9534 processor-powered Dell PowerEdge R6615 server.

We concluded our hands-on testing on November 11, 2024. During testing, we determined the appropriate hardware and software configurations and applied updates as they became available. The results in this report reflect configurations that we finalized on November 8, 2024 or earlier. Unavoidably, these configurations may not represent the latest versions available when this report appears.

Our results

To learn more about how we have calculated the wins in this report, go to http://facts.pt/calculating-and-highlighting-wins. Unless we state otherwise, we have followed the rules and principles we outline in that document.

Table 1: Results of our testing.

	Dell PowerEdge R6615 with GPU	Dell PowerEdge R6615 without GPU
Results		
Max users reached	23	9
Cut off on question response time (s)	10	10
Threshold percentage	95%	95%
Median threshold (s)	5	5
CPU AVG %UTIL @ max users	3.12	44.08
GPU AVG %UTIL @ max users	84.59	N/A
Test parameters		
LLM model	Llama-3.2-1B-Instruct-F16.gguf	Llama-3.2-1B-Instruct-F16.gguf
Corpus	airbnb	airbnb
Corpus # of listings	487,974	487974
Corpus total text size (MB)	1,952.69	1,952.69

	Dell PowerEdge R6615 with GPU	Dell PowerEdge R6615 without GPU
# of questions in conversation	5	5
Think time (s)	2.5	2.5
Think time range (s)	1	1
Ladder delay(s) (divided by number of users for each run)	5	5
User delay basis (s)	2	2
Max user delay (s)	0.1	0.1
Maximum # of tokens per prediction	128	128
Prompt context size	32,768	32,768
Threads (CPU only)	64	64
Full options for LLM service	-t 64 -c 32768 -np 100 -ngl 99 -cb --metrics	-t 64 -c 32768 -np 100 -cb --metrics
Benchmark mode	Ladder	Ladder
Begin run with # users	1	1
Increment by # users	1	1
Broker endpoint	http://100.67.144.127:8072/prompts	http://100.67.144.127:8072/prompts
Broker version	2024.11.1.56 (5 November 2024)	2024.11.1.56 (5 November 2024)
Client version	2024.11.1.56 (5 November 2024)	2024.11.1.56 (5 November 2024)

System configuration information

Table 2: Detailed information on the systems we tested.

System configuration information	Dell PowerEdge R6615 server	Dell PowerEdge R6615 client
BIOS name and version	Dell 1.8.3	Dell 1.8.3
Operating system name and version/build number	Ubuntu 22.04.5 LTS Kernel version 5.15.0-122-generic	Ubuntu 22.04.5 LTS Kernel version 5.15.0-122-generic
Date of last OS updates/patches applied	10/7/24	10/7/24
Power management policy	Performance	Performance
Processor		
Number of processors	1	1
Vendor and model	AMD EPYC™ 9534	AMD EPYC 9554
Core count (per processor)	64	64
Core frequency (GHz)	2.45	3.1
Max turbo frequency (GHz)	3.7	3.75
Stepping	1	1
Accelerators (GPU)		
Vendor and model	NVIDIA® L4	N/A
Number	1	N/A
Connection	PCIe gen4, 16 lanes	N/A
Driver	560.35.03	N/A
Memory module(s)		
Total memory in system (GB)	768	16
Number of memory modules	6	1
Vendor and model	Hynix® HMCG94AEBRA109N Hynix HMCG94AEBRA102N	Hynix HMCG78MEBRA107N
Size (GB)	64	16
Type	PC5-38400	PC5-38400
Speed (MT/s)	4,800	4,800
Speed running in the server (MT/s)	4,800	4,800
Storage controller		
Vendor and model	PERC H965i Front (Embedded)	PERC H965i Front (Embedded)
Cache size (GB)	8	8
Firmware version	8.0.0.100-00018-00022	8.0.0.0.18-81

System configuration information	Dell PowerEdge R6615 server	Dell PowerEdge R6615 client
Local storage		
Number of drives	1	1
Drive vendor and model	Dell NVMe® PM1743 RI E3.S 7.68TB	Dell Ent NVMe CM6 MU 1.6TB
Drive size (GB)	7,680	1,600
Drive information (speed, interface, type)	PCIe gen4, NVMe, SSD	PCIe gen4, NVMe, SSD
Purpose	OS, application	OS, application
Network adapter		
Vendor and model	Broadcom® BCM5720 Gigabit Ethernet	Broadcom BCM5720 Gigabit Ethernet
Number and type of ports	2 x 1GbE	2 x 1GbE
Driver version	tg3 5.15.0-122-generic	tg3 5.15.0-122-generic
Cooling fans		
Vendor and model	Dell Gold	Dell Gold
Number of cooling fans	16	16
Power supplies		
Vendor and model	Dell 06C11WA02	Dell 0C8T2PA04
Number of power supplies	2	1
Wattage of each (W)	1,400	800

How we tested

Software versions

- OS: Ubuntu Server 22.04.5 LTS (kernel version 5.15.0-122-generic)
- NVIDIA driver: 560.35.03
- CUDA toolkit: 12.6.2-1
- Docker: docker-ce 5:27.3.1-1~ubuntu.22.04~jammy
- HAproxy: 2.4.24-0ubuntu0.22.04.1
- Redis stack: redis-stack-server 7.4.0-v1
- Embedding server: michaelf34/infinity:0.0.67
- Embedding server model: sentence-transformers/msmarco-distilbert-cos-v5
- Go Lang version: 1.22.3
- PTChatterly client: 2024.11.1.56
- PTChatterly broker: 2024.11.1.56
- LLM server (CPU and GPU): llama.cpp b3849
- LLM model: Llama-3.2-1B-Instruct (official version from Huggingface)
- LLM model precision: fp16
- Nmon: 16q

Configuring the system under test (SUT)

1. Install Ubuntu Server 22.04.5 LTS, making sure that sshd is included and running.
2. Configure Ubuntu OS following the instructions in the section Configuring Ubuntu 22.04.
3. Install Docker following the instructions in the section Installing Docker on Ubuntu.
4. Install NVIDIA drivers and CUDA Toolkit following the instructions in the section Installing NVIDIA drivers.
5. Create LLM service images for both CPU and GPU:

   a. Download the tar file for this tag:

   ```
   tag=b3849 # note: change tag value to the desired version of llama.cpp
   wget https://github.com/ggerganov/llama.cpp/archive/refs/tags/"${tag}".tar.gz
   ```

   b. Untar the file and change to that directory:

   ```
   tar -xf "${tag}.tar.gz"
   cd llama.cpp-"${tag}"
   ```

   c. Get the installed version of the CUDA toolkit in X.Y.Z format:

   ```
   sudo dpkg -l cuda-toolkit*

   # this command returns, e.g., "cuda-toolkit/unknown,now 12.6.2-1 amd64 [installed]"
   # The CUDA version is then 12.6.2.
   ```

   d. Create the LLM server image with CUDA support. Note: change the CUDA_VERSION to match version installed on the system in X.Y.Z format. In some cases, the NVIDIA base image may be unavailable (the docker build returns an error[1]), then try the next lowest CUDA_VERSON.

   ```
   docker build --build-arg  CUDA_VERSION=12.6.2 -t llama-gpu-server:"${tag}" \
     -f .devops/llama-server-cuda.Dockerfile .
   ```

1 For example, "ERROR: failed to solve: nvidia/cuda:12.6.2-devel-ubuntu22.04: failed to resolve source metadata for docker.io/nvidia/cuda:12.6.2-devel-ubuntu22.04: docker.io/nvidia/cuda:12.6.2-devel-ubuntu22.04: not found"

e. Create the LLM-server image with CPU-only support:

```
docker build -t llama-cpu-server:"${tag}" -f .devops/llama-server.Dockerfile .
unset tag
```

6. Install Redis Stack, the vector database:

a. Uninstall the non-vector-db Redis server (ignore errors if it isn't present):

```
sudo apt remove redis redis-server redis-tools
```

b. Add Redis Stack Server repo to the system:

```
curl -fsSL https://packages.redis.io/gpg | \
  sudo gpg --dearmor -o /usr/share/keyrings/redis-archive-keyring.gpg
sudo chmod 644 /usr/share/keyrings/redis-archive-keyring.gpg
echo "deb [signed-by=/usr/share/keyrings/redis-archive-keyring.gpg] https://packages.redis.io/
deb $(lsb_release -cs) main" | \
  sudo tee /etc/apt/sources.list.d/redis.list
```

c. Install Redis "stack server" and the Redis CLI, for possible debugging:

```
sudo apt update
sudo apt install redis-stack-server redis-tools
```

d. Configure the instance to allow anonymous external connections. Replace ##IPADDR## with the external IP address of the server.

```
(echo '# start of mods for ptchatterly'
echo 'protected-mode no'
echo 'bind 127.0.0.1 ##IPADDR##') | \
  sudo tee -a /etc/redis-stack.conf
```

e. Restart the service and enable it to start when the system does:

```
sudo systemctl enable redis-stack-server.service
sudo systemctl restart redis-stack-server.service
```

f. Configure the OS to permit memory over-commit:

```
echo 'vm.overcommit_memory = 1' | sudo tee -a /etc/sysctl.conf
sudo sysctl -p
```

7. Prepare the LLM model for the LLM service:

```
# create the directory to hold the model(s)
sudo mkdir -p /usr/local/llama.cpp/models/PT/
sudo chmod 1777 /usr/local/llama.cpp/models/PT
# prepare virtual pythion environment; python and git may have already been installed
sudo apt install python3 python3-venv git
python3 -m venv .env
. .env/bin/activate
# install prerequisites packages
pip install --upgrade huggingface_hub[hf_transfer] hf_transfer
git config --global credential.helper store
```

a. Download the model:

```
cd /usr/local/llama.cpp/models/PT
huggingface-cli login --token <HUGGING_FACE_TOKEN> --add-to-git-credential
HF_HUB_ENABLE_HF_TRANSFER=1 huggingface-cli download meta-llama/Llama-3.2-1B-Instruct --local-
dir ./Llama-3.2-1B-Instruct
```

b. Convert the model to GGUF format with the datatype fp16. The converted model will be in directory /usr/local/llama.cpp/models/PT/Llama-3.2-1B

```
docker run --rm -v /usr/local/llama.cpp/models/PT:/models ghcr.io/ggerganov/llama.cpp:full \
  --convert /models/Llama-3.2-1B-Instruct/ --outtype fp16
```

8. Install the nmon resource monitoring tool on the SUT. The resulting binary file(s) can be copied to the remaining systems:

```
# Remove any preinstalled versions of nmon
sudo apt remove -y nmon
# Build requirements:
#  ncurses-dev
#  NVIDIA GPU drivers for SUTs with GPUs (already installed)
#  gcc, make, and wget (already installed)
sudo apt install ncurses-dev
# Download source files
mkdir nmon_build
cd nmon_build
wget 'https://sourceforge.net/projects/nmon/files/lmon16q.c/download' -O lmon.c
wget 'https://sourceforge.net/projects/nmon/files/makefile/download' -O Makefile
```

- For SUTs with GPUs, build and install nmon with GPU support:

```
# patch the Makefile so that it finds and uses the NVIDIA GPU management library

make gpu
sudo install nmon_X86_Ubuntu22_16q_gpu /usr/local/sbin/nmon
```

- For SUTs without GPUs, build and install nmon without GPU support:

```
make
sudo install nmon_X86_Ubuntu22_16q /usr/local/sbin/nmon
```

9. Install nmonchart, the nmon parser and chart creator, on the client system:

```
# Set up nmonchart in a convenient directory on the client system
# The script uses the Korn shell
sudo apt install ksh
wget 'https://sourceforge.net/projects/nmon/files/nmonchart42.tar/download' -O nmonchart42.tar
tar -xvf nmonchart42.tar ./nmonchart
# #FIXME# does this fix apply to CPU-only systems? What about tests w/ and w/o GPUs on the samee SUT?
cp nmonchart nmonchat-orig
sed -i 'sZ/,0,0/Z/,0,0,0,0/Z' nmonchart
sudo install nmonchart /usr/local/sbin/nmonchart
```

10. Perform the following steps:

a. Copy service start scripts to the SUT:

```
start_llama_all.sh, start_redis.sh, start_infinity.sh.
```

b. Start the LLM service:

```
bash start_llama_all.sh
```

c. Start embedding service:

```
bash start_infinity.sh
```

d. Start Redis vector database:

```
bash start_redis.sh
```

11. Prepare SUT to ingest Airbnb data:

a. Create a new directory on the SUT:

```
mkdir ingest
cd ingest
```

b. Create and activate a new virtual Python environment:

```
sudo apt install python3-venv
python3 -m venv .
PATH="~/.local/bin:$PATH"
. ./bin/activate
```

12. Copy the following files to this directory: property listings file dataset, `AirbnbProps-20240830.json.gz`, and the Python ingestion script, `ingestAirBnB.py`

a. Create a symbolic link to property listings file dataset:

```
ln -s AirbnbProps-20240830.json.gz AirbnbProps.json.gz
```

b. Add Python packages:

```
pip install -U redis sentence-transformers pandas
```

c. Ingest the data:

```
time python3 ingestAirBnB.py
```

Configuring the Associated client / test harness server

1. Install Ubuntu Server 22.04.5 LTS, making sure that sshd is included and running.
2. Configure Ubuntu OS following the instructions in the section Configuring Ubuntu 22.04.
3. Install Docker following the instructions in the section Installing Docker on Ubuntu.
4. Install HAProxy:

```
sudo apt update
sudo apt install haproxy
sudo systemctl disable haproxy
```

5. Install the PTChatterly broker and client.

Configuring Ubuntu 22.04

1. After installation, perform these configuration steps.
2. We assume the login for the non-root user is ptuser.
3. Enable password-less sudo:

```
echo "$USER ALL=(ALL:ALL) NOPASSWD: ALL" | sudo tee /etc/sudoers.d/$USER
sudo chmod 640 /etc/sudoers.d/$USER
```

4. Set time zone; e. g.,

```
sudo timedatectl set-timezone America/New_York
```

5. Extend the root LVM filesystem, if necessary:

```
sudo lvextend -r -l +100%FREE /dev/ubuntu-vg/ubuntu-lv
```

6. Disable unattended package updates:

```
sudo systemctl stop unattended-upgrades.service
sudo systemctl disable unattended-upgrades.service
```

7. Modify the value of the unattended-upgrade variable to 0 in file /etc/apt/apt.conf.d/20auto-upgrades:

```
APT::Periodic::Update-Package-Lists "1";
APT::Periodic::Unattended-Upgrade "0";
```

8. Install the latest updates:

```
sudo apt update
sudo apt upgrade
```

9. Install standard Ubuntu packages you will need in subsequent steps:

```
sudo apt install ca-certificates curl wget lsb-release sysstat smartmontools vim nmon numactl
```

10. Reboot the system:

```
sudo shutdown -r now
```

Installing Docker on Ubuntu 22.04

1. Remove any previous Docker packages:

```
for pkg in docker.io docker-doc docker-compose docker-compose-v2 \
    podman-docker containerd runc; do \
  sudo apt remove $pkg
done
```

2. Add Docker's official GPG key:

```
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
```

3. Add the Docker repository to the system:

```
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.
docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt update
```

4. Install Docker CE:

```
sudo apt update
sudo apt install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
```

5. Add current user to Docker permissions group, and add docker group to the user's current session:

```
sudo usermod -aG docker $USER
newgrp docker
```

6. Confirm Docker function under non-root user:

```
docker run --rm hello-world
```

7. Enable Docker services:

```
sudo systemctl enable --now docker.service
sudo systemctl enable --now containerd.service
```

Installing NVIDIA drivers and CUDA Toolkit

1. Add the NVIDIA repo to the system:

```
# Notes:
# 1. replace ubuntu2204 in the URL with the version of your distro if not 22.04.
# 2. Replace the CUDA toolkit version and NVIDIA driver version, if necessary

wget https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2204/x86_64/cuda-
keyring_1.1-1_all.deb
sudo dpkg -i cuda-keyring_1.1-1_all.deb
sudo apt update

# Note: adjust the versions of the drivers and CUDA as necessary
sudo apt install -y nvidia-headless-560
sudo apt install -y nvidia-utils-560 nvidia-container-toolkit nvtop
sudo apt install -y cuda-toolkit-12-6
```

2. Update your session's environment so you can use the CUDA tools. Add the following lines to your .bashrc profile for future sessions:

```
export PATH=/usr/local/cuda/bin${PATH:+:${PATH}}
export LD_LIBRARY_PATH=/usr/local/cuda/lib64${LD_LIBRARY_PATH:+:${LD_LIBRARY_PATH}}
```

3. Reboot the system (necessary because the kernel's default module list has been updated):

```
sudo shutdown -r now
```

4. Validate GPU functionality (optional):

```
docker run --rm --gpus all nvcr.io/nvidia/k8s/cuda-sample:nbody nbody -gpu -benchmark
```

Read the report at https://facts.pt/MxYZe83 ▶

This project was commissioned by Dell Technologies.