A Principled Technologies report: Hands-on testing. Real-world results.

Achieve near-bare-metal inference throughput for image classification workloads with the Dell PowerEdge R7525 server using virtual GPUs

Plus, virtualizing servers with VMware vSphere 7.0 Update 3 and vGPUs could help simplify machine learning workload management

Minimize performance degradation from virtualization

Achieve 97.5% of the image classification performance of a bare-metal configuration

Simplify ML workload management

Admins can ensure uptime and resource allocation through virtual management

Virtualization has changed the world of computing, but every hypervisor adds some amount of computing overhead compared to a bare-metal environment. While this overhead may not present an issue for many workloads, businesses may well feel those effects on compute-heavy workloads, such as machine learning (ML). With VMware® vSphere® 7.0 Update 3, Dell™ PowerEdge™ R7525 servers powered by AMD EPYC™ processors offer a platform that can minimize performance loss with virtualized GPUs (vGPUs) while delivering the ease-of-management, redundancy, reliability, mobility, and security advantages that virtualization can provide.

In our data center, we tested ML-based image classification inference throughput on bare-metal and virtual configurations of a Dell PowerEdge R7525 with two AMD EPYC 7543 processors and an NVIDIA® A100 Tensor Core GPU. When running the MLPerf™ ResNet50 workload, the bare-metal configuration processed 28,130 queries per second. The virtual configuration, with VMware vSphere 7.0 Update 3 and a single guest VM, processed 27,435 queries per second—97.5 percent of the bare-metal configuration's image classification performance. Either configuration of the PowerEdge R7525 could serve as the backbone in image-classification workloads, delivering strong performance as a bare-metal or virtual solution. Virtualizing the PowerEdge R7525 with NVIDIA vGPU software and vSphere 7 Update 3 could enable IT admins to run multiple virtual machines (VMs) on the same physical GPU, making it easier to dial in GPU-level performance as needed.

Achieve near-bare-metal inference throughput for image classification workloads with the Dell PowerEdge R7525 server using virtual GPUs July 2022

The science behind the report:

Achieve near-bare-metal inference throughput for image classification workloads with the Dell PowerEdge R7525 server using virtual GPUs

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 Achieve near-bare-metal inference throughout for image classification workloads with the Dell PowerEdge R7525 server using virtual GPUs.

We concluded our hands-on testing on May 26, 2022. 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 April 25, 2022 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

	Target QPS	Achieved QPS	Mean latency (ns)	Performance achieved compared to baremetal
vGPU	27440	27435.1	7374352	97.5
Baremetal	28135	28130.7	7958499	-

System configuration information

Table 2: Detailed information on the system we tested.

System configuration information	Dell™ PowerEdge™ R7525
BIOS name and version	Dell 2.6.6
Non-default BIOS settings	Global SRIOV enabled, Max Performance enabled, 8G decoding enabled, UEFI boot
Operating system name and version/build number	Ubuntu 20.04, VMware® vSphere® 7.0 Update 3
Date of last OS updates/patches applied	4/25/22
Power management policy	Performance
Processor	
Number of processors	2
Vendor and model	AMD EPYC™ 7543
Core count (per processor)	32
Core frequency (GHz)	2.80 (3.90 Boost)
Stepping	1
Memory module(s)	
Total memory in system (GB)	512
Number of memory modules	16
Vendor and model	Micron® 36ASF4G72PZ-3G2E7
Size (GB)	32
Type	PC4-3200
Speed (MHz)	3,200
Speed running in the server (MHz)	3,200
Storage controller	
Vendor and model	Dell BOSS-S1 M.2 SSD
Cache size (GB)	N/A
Firmware version	2.5.13.3024
Local storage	
Number of drives	2
Drive vendor and model	Micron MTFDDAV480TCB
Drive size (GB)	480
Drive information (speed, interface, type)	M.2 SSD
Network adapter	
Vendor and model	Broadcom® BCM5720
Number and type of ports	4 x 1GbE
Driver version	1.39

System configuration information	Dell™ PowerEdge™ R7525
Cooling fans	
Vendor and model	Dell HPR Gold
Number of cooling fans	12
Power supplies	
Vendor and model	Dell 01CW9G
Number of power supplies	2
Wattage of each (W)	1,400

Table 3: Detailed configuration information for the GPU.

System configuration information	NVIDIA A100
Firmware revision	92.00.25.00.08
PCIe width	16x
GPU memory (GB)	40
Non-default settings used	ECC enabled

How we tested

Testing overview

We tested two configurations: one bare metal, and the other virtualized. The virtual environment used VMware vSphere 7.0 Update 3 as the hypervisor and Ubuntu 20.04 as the guest OS. The bare-metal configuration used Ubuntu 20.04. Both installations used a Dell BOSS-S1 SSD card to boot the OS, and both configurations of Ubuntu 20.04 were identical except for changing the PCIe ID listed in the MLPerf ResNet50 config files (bare-metal GPU presents a different ID than the virtualized NVIDIA GRID device). We used a single NVIDIA A100 GPU in both environments.

Creating the bare-metal and virtual environments

This section contains the steps we took to create our bare-metal and virtual test environments.

Configuring the server

1. We made sure the Dell PowerEdge R7525 had the proper GPU enablement hardware installed, and latest BIOS/firmware.
2. In the server BIOS settings, ensure that:
    - SRIOV is globally enabled
    - 8G decoding is enabled
    - UEFI boot is enabled
    - Use the Max Performance server profile.

Installing and configuring VMware vSphere 7.0 Update 3

Use these steps to install the hypervisor, configure the NVIDIA vGPU technology, and create a VM. If testing on the bare-metal environment, skip this section and proceed to Installing the OS.

1. Boot the server to the VMware vSphere 7.0 Update 3 installation media. We used the iDRAC virtual media attachment option to mount the ISO file.
2. Press Enter, and press F11 to accept the license agreement and continue.
3. Select the BOSS SSD RAID volume, and press Enter.
4. Select US Default, and press Enter.
5. Enter a password, and press Enter.
6. To begin the install, press F11.
7. Once the install completes, use the Troubleshooting menu to enable remote shell and SSH service.
8. SSH to the host, and run the following:

```
esxcli graphics host set --default-type SharedPassthru
```

9. Reboot the host.
10. Download the NVAIE Host vGPU driver for VMware vSphere from the NVAIE portal to the ESXi host.
11. Install the driver by putting the host into Maintenance Mode, and running:

```
esxcli software vib install -v <full path of .vib file>
```

12. Take the host out of Maintenance Mode, and verify the install worked by running `nvidia-smi`.

Creating the VM

1. Attach the ESXi host to an existing VMware vCenter.
2. Create a VM in VMware vSphere with the following attributes:
    - 64 vCPU
    - 128 GB memory
    - 100% of memory reserved
    - New PCIe device: NVIDIA GRID A100

Installing the OS

Use the following steps on the test environment (on the server if testing bare-metal, and on the VM if testing virtualized).

1. Boot the machine to the Ubuntu Server 20.04 LTS installation media.
2. When prompted, select Install Ubuntu.
3. Select the desired language, and click Done.
4. Choose a keyboard layout, and click Done.
5. At the Network Connections screen, click Done.
6. At the Configure Proxy screen, click Done.
7. At the Configure Ubuntu Archive Mirror screen, click Done.
8. Select Use an entire disk, and click Done.
9. Click Continue.
10. Enter user account details, and click Done.
11. Enable OpenSSH Server install, and click Done.
12. At the installation summary screen, click Done.
13. When the installation finishes, unmount the installation media and reboot the machine.

Configuring Ubuntu Server 20.04 LTS

1. Log in as the user created in the previous section.
2. Install the latest update packages and reboot the VM.

```
sudo apt-get update
sudo apt-get upgrade -y
sudo reboot
```

3. Set the time zone on the VM.

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

4. Install additional tools:

```
sudo apt-get install -y nmon dkms build-essential
```

Installing the NVIDIA driver and runtime container

1. For the bare-metal configuration, download the NVIDIA Data Center Driver for Linux x64. For the virtualized configuration, use the NVAIE Guest vGPU driver. If on virtualized, follow these instructions after installing the driver to license the vGPU: https://docs.nvidia.com/grid/13.0/grid-licensing-user-guide/index.html.
2. Install the package with:

```
chmod +x <path to driver installation package>
dpkg -i <path to driver installation package>
```

3. Run `nvidia-smi to ensure the driver installed correctly.`
4. Reboot the system.
5. Add the Docker GPG key and install Docker:

```
sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/
keyrings/docker.gpg
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin
```

6. Set up the NVIDIA package repository and GPG key:

```
distribution=$(. /etc/os-release;echo $ID$VERSION_ID) \
&& curl -fsSL https://nvidia.github.io/libnvidia-container/gpgkey | sudo gpg --dearmor -o /usr/
share/keyrings/nvidia-container-toolkit-keyring.gpg \
&& curl -s -L https://nvidia.github.io/libnvidia-container/$distribution/libnvidia-
container.list | \
sed 's#deb https://#deb [signed-by=/usr/share/keyrings/nvidia-container-toolkit-keyring.gpg]
https://#g' | \
sudo tee /etc/apt/sources.list.d/nvidia-container-toolkit.list
```

7. Update the apt listing, install the NVIDIA container package, and restart Docker:

```
apt-get update
apt-get install -y nvidia-docker2
systemctl restart docker
```

8. Test the NVIDIA Docker functionality with:

```
docker run --rm --gpus all nvidia/cuda:11.0.3-base-ubuntu20.04 nvidia-smi
```

Setting up the machine learning benchmark

1. Clone the MLPerf code base to your desired location. We used /opt/.

```
git clone https://github.com/mlcommons/inference_results_v1.0
```

2. Edit configs/resnet50/Server/config.json to include the system under test:

```
{
  "benchmark": "resnet50",
  "default": {
    "active_sms": 100,
    "input_dtype": "int8",
    "input_format": "linear",
    "map_path": "data_maps/imagenet/val_map.txt",
    "precision": "int8",
    "tensor_path": "${PREPROCESSED_DATA_DIR}/imagenet/ResNet50/int8_linear",
    "use_deque_limit": true
  },
  "scenario": "Server",
  "R7525xa_GRID-A100-40Cx1": {
    "config_ver": {},
    "deque_timeout_us": 2000,
    "deque_timeout_usec": 2000,
    "use_cuda_thread_per_device": true,
    "use_graphs": true,
    "gpu_batch_size": 64,
    "gpu_copy_streams": 4,
    "gpu_inference_streams": 3,
    "server_target_qps": <target QPS depending on desired load>
  }
}
```

3. Add the system to code/common/system_list.py:

```
R7525_GRID_A100_40C = SystemClass("R7525xa_GRID-A100-40C", ["GRID A100-40C"], [], A
rchitecture.Ampere, [1] )
```

4. Follow the instructions at https://github.com/mlcommons/training/tree/master/image_classification to register for, download, and pre-process the ILSVRC2012 dataset.

Running the tests

In this section, we list the steps to build the test container and run the test. We also captured performance metrics with nvidia-smi, esxtop, and nmon.

1. Edit config/ResNet50/Server/config.json to reflect the desired target QPS.
2. Run the benchmark container build script:

```
sudo su
/opt/mlperf-runner/run-mlperf-container.sh -i mlperf-inference:dell-latest -h mlperf-inference-
userv1.0 -n mlperf-inference-user -o mlperf-inference-v1.0-dellemc  --build 2>&1 | tee /opt/mlperf-
runner/build-log.log
```

3. Run the benchmark test script:

```
/opt/mlperf-runner/run-mlperf-container.sh -i mlperf-inference-v1.0-dellemc:latest --run-server -t
--server_target_latency_ns 15000000 -- 2>&1 | tee /opt/mlperf-runner/run-log.log
```

Test scripts

run-mlperf-container.sh

```
#!/bin/bash
#! -------------------------------------------------------
#!        !!! Require root privileges !!!
#! -------------------------------------------------------
if [[ $UID -ne 0 ]]; then
  echo "Script started as '`whoami`' instead of 'root'. Restarting script as 'root' user using sudo."
  sudo bash $0 "$@"
  exit $?
fi

echo "SCRIPT:     $0"
echo "ARGUMENTS:  $@"
echo "WORKING DIR: `pwd`"
echo "RUNNING AS:  '`whoami`'"
echo "ENVIRONMENT:"
env | egrep -v '^(PATH|LS_COLORS)' | sed 's/^/  | /' | column -ntxs '='
echo "PATH"
echo "$PATH" | sed 's/:/\n/g' | sed 's/^/  | /'

# -------------------------------------------------------
# Argument initialization
# -------------------------------------------------------
IMAGE_NAME=
CONTAINER_NAME=
CONTAINER_HOSTNAME=
OUTPUT_IMAGE=
declare -a EXTRA_ARGS
declare -a TEST_EXTRA_ARGS
declare -a CMD

# outer directories (base)
DATA_DIR=/data/mlperf/ilsvrc2012
CODE_DIR=/opt/mlperf-inference-v1.0/code
USER_DIR=/opt/mlperf-inference-v1.0/user
LOG_DIR=/var/log/mlperf

# -------------------------------------------------------
# Error handling
# -------------------------------------------------------
function errexit(){
  echo "Error: $@" 1>&2
  echo "Aborting..." 1>&2
  exit 1
}

# -------------------------------------------------------
# Parse Arguments
# -------------------------------------------------------
while [ $# -gt 0 ]; do
  arg="$1"
  shift
  case "$arg" in
    -i|--image)        IMAGE_NAME=$1;         shift ;;
    -h|--hostname)     CONTAINER_HOSTNAME=$1; shift ;;
    -n|--name)         CONTAINER_NAME=$1;     shift ;;
    -o|--output-image) OUTPUT_IMAGE=$1;       shift ;;
    -d|--data-dir)     DATA_DIR=$1;           shift ;;
    -c|--code-dir)     CODE_DIR=$1;           shift ;;
    -u|--user-dir)     USER_DIR=$1;           shift ;;
    --log-dir)         LOG_DIR="$1"           shift ;;
    -x|--extra-args)
      # Consume remaining arguments up to a '--' argument which resumes normal parsing.
      while [ $# -gt 0 ]; do
        xarg="$1"
        shift
        if [ "$xarg" == "--" ]; then
          break
        else
          EXTRA_ARGS+=("${xarg}")
        fi
      done
    ;;
    -t|--test-args)
      # Consume remaining arguments up to a '--' argument which resumes normal parsing.
      while [ $# -gt 0 ]; do
        arg="$1"
        shift
        if [ "$arg" == "--" ]; then
          break
        else
          TEST_EXTRA_ARGS+=("${arg}")
        fi
      done
    ;;
    --|--cmd)
      # start of command. Consume remaining arguments
      while [ $# -gt 0 ]; do
        CMD+=("${1}")
        shift
      done
    ;;
    -b|--build)
      # CMD=(bash -c 'export DEBIAN_FRONTEND=noninteractive && apt-get install -y tree jq htop && make 
download_model BENCHMARKS=resnet50 && make build && make generate_engines RUN_ARGS="--
benchmarks=resnet50 --scenarios=Offline,Server,SingleStream,MultiStream --config_ver=default"')
      # CMD=(bash -c 'export DEBIAN_FRONTEND=noninteractive && apt-get install -y tree jq htop && make 
download_model BENCHMARKS=resnet50 && make build && make generate_engines RUN_ARGS="--
benchmarks=resnet50 --scenarios=Offline,Server --config_ver=default"')
      CMD=(bash -c 'export DEBIAN_FRONTEND=noninteractive && apt-get install -y tree jq htop && make 
--debug -j download_model BENCHMARKS=resnet50 && make --debug -j build && make --debug -j generate_
engines RUN_ARGS="--benchmarks=resnet50 --scenarios=Offline,Server --config_ver=default"')
    ;;
    --run-server)       RUN=yes; SCENARIO=Server        ;;
    --run-offline)     RUN=yes; SCENARIO=Offline       ;;
    # --run-singlestream) RUN=yes; SCENARIO=SingleStream  ;;
    # --run-multistream)  RUN=yes; SCENARIO=MultiStream   ;;
    --bash)
                        CMD=(bash);
                        EXTRA_ARGS+=("-it")
                        CONTAINER_HOSTNAME=abani-mlperf-bash
                        CONTAINER_NAME=abani-mlperf-bash
                        OUTPUT_IMAGE=
    ;;
    --remove|--rm)      EXTRA_ARGS+=("--rm"); OUTPUT_IMAGE= ;;
    *)
      # unrecognized option, must be start of command. Consume remaining arguments
      CMD+=("${arg}")
      while [ $# -gt 0 ]; do
        CMD+=("${1}")
        shift
      done
    ;;
  esac
done

LOG_DIR_INNER=/mlperf-logs
mkdir -p "${LOG_DIR}"

if [ "$RUN" == yes ]; then
  CMD=(python3 code/main.py --benchmarks=resnet50 --scenarios=$SCENARIO --config_ver=default --test_
mode=PerformanceOnly --action=run_harness --log_dir=${LOG_DIR_INNER} )
  CMD+=( "${TEST_EXTRA_ARGS[@]}" )
  EXTRA_ARGS+=("-e" "PREPROCESSED_DATA_DIR=/scratch/preprocessed_data" "--rm")
  CONTAINER_HOSTNAME=abani-mlperf
  CONTAINER_NAME=abani-mlperf
  OUTPUT_IMAGE=
fi

# -------------------------------------------------------
# Argument checks and defaults
# -------------------------------------------------------
[ -z "${CMD}" ] && errexit "Missing command! Note: Specify with --cmd X Y Z, -- X Y Z, or just X Y Z at
the end of all other options."
if [ -z "${IMAGE_NAME}" ]; then IMAGE_NAME=mlperf-inference:dell-latest; fi
if [ -z "${CONTAINER_NAME}" ]; then CONTAINER_NAME=${IMAGE_NAME/:*/}; fi
if [ -z "${CONTAINER_HOSTNAME}" ]; then CONTAINER_HOSTNAME=${CONTAINER_NAME}; fi

[ ! -d "${DATA_DIR}" ] && errexit "Missing data directory! Note: Specify with --data-dir DIRECTORY."
[ ! -d "${CODE_DIR}" ] && errexit "Missing code directory! Note: Specify with --code-dir DIRECTORY."
[ ! -d "${USER_DIR}" ] && errexit "Missing user directory! Note: Specify with --user-dir DIRECTORY."



# -------------------------------------------------------
# Derived Arguments and constants
# -------------------------------------------------------
# outer directories (subdirs)
WORK_DIR=${CODE_DIR}/closed/DellEMC
MAPS_DIR=${CODE_DIR}/closed/NVIDIA/data_maps/imagenet
#SCRIPTS_DIR=${CODE_DIR}/closed/NVIDIA/scripts

# inner directories
USER_DIR_INNER=/mnt/user
WORK_DIR_INNER=/work
MAPS_DIR_INNER=${WORK_DIR_INNER}/data_maps/imagenet
#SCRIPTS_DIR_INNER=${WORK_DIR_INNER}/scripts
SCRATCH_DIR_INNER=/scratch
DATA_DIR_INNER=${SCRATCH_DIR_INNER}/preprocessed_data

# -------------------------------------------------------
# Derived Docker invocation arguments
# -------------------------------------------------------
VOLUME_ARGS="-v ${LOG_DIR}:${LOG_DIR_INNER} -v ${DATA_DIR}:${DATA_DIR_INNER} -v ${WORK_DIR}:${WORK_DIR_
INNER} -v ${USER_DIR}:${USER_DIR_INNER} -v ${MAPS_DIR}/cal_map.txt:${MAPS_DIR_INNER}/cal_map.txt:ro -v
${MAPS_DIR}/val_map.txt:${MAPS_DIR_INNER}/val_map.txt:ro -v /etc/timezone:/etc/timezone:ro -v /etc/
localtime:/etc/localtime:ro"
# -v ${SCRIPTS_DIR}:${SCRIPTS_DIR_INNER}
GPU_ARGS="--gpus=all"
SECURITY_ARGS="--security-opt apparmor=unconfined --security-opt seccomp=unconfined --cap-add SYS_ADMIN"
ENVIRONMENT_ARGS="-w ${WORK_DIR_INNER} -e MLPERF_SCRATCH_PATH=${SCRATCH_DIR_INNER} -e NVIDIA_MIG_
CONFIG_DEVICES=all"
DEVICE_ARGS="--device /dev/fuse"
HOST_ARGS="-h ${CONTAINER_HOSTNAME} --add-host ${CONTAINER_HOSTNAME}:127.0.0.1"
NET_ARGS="--net host"
DNS_ARGS="--dns 172.16.100.250 --dns 10.41.0.10 --dns 10.41.0.11 --dns 10.41.0.12 --dns-search abani.
local --dns-search vsphere.local --dns-search principledtech.com"
CONTAINER_ARGS="--name ${CONTAINER_NAME}"

function run(){
  echo "RUNNING COMMAND: $@"
  "${@}"
}

# -------------------------------------------------------
# Docker run function
# -------------------------------------------------------
function run_container(){
  run                 \
  docker run          \
  ${VOLUME_ARGS}      \
  ${GPU_ARGS}         \
  ${SECURITY_ARGS}    \
  ${ENVIRONMENT_ARGS} \
  ${DEVICE_ARGS}      \
  ${HOST_ARGS}        \
  ${NET_ARGS}         \
  ${DNS_ARGS}         \
  ${CONTAINER_ARGS}   \
  "${EXTRA_ARGS[@]}"  \
  ${IMAGE_NAME} \
  "${CMD[@]}"
}

# ---------------------------------------------------------
# Docker commit function
# ---------------------------------------------------------
function commit_container(){
  if [ -z "${OUTPUT_IMAGE}" ]; then
    echo "Not committing container to image as no output was specified."
  else
    run docker stop "${CONTAINER_NAME}";
    run docker commit "${CONTAINER_NAME}" "${OUTPUT_IMAGE}" &&
    run docker rm "${CONTAINER_NAME}"
  fi
}

# ---------------------------------------------------------
# main entry point
# ---------------------------------------------------------
# Run the specified command in the specified mlperf-based
# container, optionally commiting the container as a new image
run_container && commit_container || errexit "Failed to run container or commit container to image."
```

Read the report at https://facts.pt/Y9ecZ6o ▶

This project was commissioned by Dell Technologies.

DISCLAIMER OF WARRANTIES; LIMITATION OF LIABILITY:
Principled Technologies, Inc. has made reasonable efforts to ensure the accuracy and validity of its testing, however, Principled Technologies, Inc. specifically disclaims any warranty, expressed or implied, relating to the test results and analysis, their accuracy, completeness or quality, including any implied warranty of fitness for any particular purpose. All persons or entities relying on the results of any testing do so at their own risk, and agree that Principled Technologies, Inc., its employees and its subcontractors shall have no liability whatsoever from any claim of loss or damage on account of any alleged error or defect in any testing procedure or result.

In no event shall Principled Technologies, Inc. be liable for indirect, special, incidental, or consequential damages in connection with its testing, even if advised of the possibility of such damages. In no event shall Principled Technologies, Inc.'s liability, including for direct damages, exceed the amounts paid in connection with Principled Technologies, Inc.'s testing. Customer's sole and exclusive remedies are as set forth herein.