



The science behind the report:

AWS EC2 M6i instances with 3rd Gen Intel Xeon Scalable processors accelerated image classification for machine learning workloads

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 [AWS EC2 M6i instances with 3rd Gen Intel Xeon Scalable processors accelerated image classification for machine learning workloads](#).

We concluded our hands-on testing on December 22, 2021. 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 December 22, 2021 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: ResNet50 results for 16vCPU instances: M6i vs. M6a. Table 2: ResNet50 results for 96vCPU instances: M6i vs. M6a.

Instance	Batch size	Precision	FPS	Times win
m6a.4xlarge	1	fp32	56.8	1
m6i.4xlarge	1	fp32	102.0	1.79
m6a.4xlarge	1	int8	150.4	1
m6i.4xlarge	1	int8	225.2	1.49
m6a.4xlarge	128	fp32	62	1
m6i.4xlarge	128	fp32	138.4	2.23
m6a.4xlarge	128	int8	174.1	1
m6i.4xlarge	128	int8	435.2	2.49

Instance	Batch size	Precision	FPS	Times win
m6a.24xlarge	1	fp32	271.0	1
m6i.24xlarge	1	fp32	626.2	2.31
m6a.24xlarge	1	int8	781.5	1
m6i.24xlarge	1	int8	1403.4	1.79
m6a.24xlarge	128	fp32	282.8	1
m6i.24xlarge	128	fp32	780.9	2.76
m6a.24xlarge	128	int8	898.4	1
m6i.24xlarge	128	int8	2649.6	2.94

Table 3: ResNet50 results for 16vCPU instances: M6i vs. M5n. Table 4: ResNet50 results for 96vCPU instances: M6i vs. M5n.

Instance	Batch size	Precision	FPS	Times win
m5n.4xlarge	1	int8	199.1	1
m6i.4xlarge	1	int8	225.2	1.13
m5n.4xlarge	128	int8	365.7	1
m6i.4xlarge	128	int8	435.2	1.19

Instance	Batch size	Precision	FPS	Times win
m5n.24xlarge	1	int8	1141.3	1
m6i.24xlarge	1	int8	1403.4	1.22
m5n.24xlarge	128	int8	2178.7	1
m6i.24xlarge	128	int8	2649.6	1.21

System configuration information

Table 5: Detailed information on the M5n instances we tested.

System configuration information	M5n (16 vCPU)	M5n (96 vCPU)
Tested by	Principled Technologies	Principled Technologies
Test date	FP32,BS=1: 22/12/2021 FP32,BS=128: 22/12/2021 INT8,BS=1: 21/12/2021 INT8,BS=128: 21/12/2021	FP32,BS=1: 22/12/2021 FP32,BS=128: 22/12/2021 INT8,BS=1: 21/12/2021 INT8,BS=128: 21/12/2021
CSP / Region	AWS EC2 – US East 1	AWS EC2 – US East 1
Workload & version	Model Zoo for Intel Architectures v2.4.0: resnet50v1_5	Model Zoo for Intel Architectures v2.4.0: resnet50v1_5
WL specific parameters	<i>See table 9: tuning profile table</i> FP32,BS=1: Profile E FP32,BS=128: Profile D INT8,BS=1: Profile F INT8,BS=128: Profile D	<i>See table 9: tuning profile table</i> FP32,BS=1: Profile E FP32,BS=128: Profile D INT8,BS=1: Profile F INT8,BS=128: Profile D
Iterations and result choice	3 runs, median	3 runs, median
Server platform	m5n.4xlarge	m5n.24xlarge
BIOS name and version	Amazon EC2 1.0 10/16/2017	Amazon EC2 1.0 10/16/2017
Operating system name and version/build number	Ubuntu 20.04 5.4.0-1045-aws	Ubuntu 20.04 5.4.0-1045-aws
Date of last OS updates/ patches applied	FP32,BS=1: 22/12/2021 FP32,BS=128: 22/12/2021 INT8,BS=1: 21/12/2021 INT8,BS=128: 21/12/2021	FP32,BS=1: 22/12/2021 FP32,BS=128: 22/12/2021 INT8,BS=1: 21/12/2021 INT8,BS=128: 21/12/2021
Processor		
Number of processors	1	1
Vendor and model	Intel® Xeon® 8259CL	Intel Xeon 8259CL
Core count (per processor)	24	24
Core frequency (GHz)	2.50	2.50
Stepping	7	7
Hyper-threading	Yes	Yes
Turbo	Yes	Yes
Number of vCPU per VM	16	96
Memory module(s)		
Total memory in system (GB)	64	384
NVMe memory present?	No	No
Total memory (DDR+NVMe RAM)	64	384
General HW		
Storage: NW or Direct Att / Instance	Direct Att / Instance	Direct Att / Instance

System configuration information	M5n (16 vCPU)	M5n (96 vCPU)
Local storage		
OS		
Number of drives	1	1
Drive size (GB)	30	30
Drive information (speed, interface, type)	Standard SSD (GP2)	Standard SSD (GP2)
Data drive		
Number of drives	1	1
Drive size (GB)	200	200
Drive information (speed, interface, type)	Standard SSD (GP2)	Standard SSD (GP2)
Network adapter		
Vendor and model	Amazon Elastic Network Adapter	Amazon Elastic Network Adapter
Number and type of ports	1x 25Gb	1x 100Gb

Table 6: Detailed information on the M6a instances we tested.

System configuration information	M6a (16 vCPU)	M6a (96 vCPU)
Tested by	Principled Technologies	Principled Technologies
Test date	FP32,BS=1: 22/12/2021 FP32,BS=128: 21/12/2021 INT8,BS=1: 21/12/2021 INT8,BS=128: 21/12/2021	FP32,BS=1: 22/12/2021 FP32,BS=128: 22/12/2021 INT8,BS=1: 21/12/2021 INT8,BS=128: 21/12/2021
CSP / Region	AWS EC2 – US East 1	AWS EC2 – US East 1
Workload & version	Model Zoo for Intel Architectures v2.4.0: resnet50v1_5	Model Zoo for Intel Architectures v2.4.0: resnet50v1_5
WL specific parameters	<i>See table 9: tuning profile table</i> FP32,BS=1: Profile E FP32,BS=128: Profile F INT8,BS=1: Profile E INT8,BS=128: Profile D	<i>See table 9: tuning profile table</i> FP32,BS=1: Profile E FP32,BS=128: Profile D INT8,BS=1: Profile E INT8,BS=128: Profile D
Iterations and result choice	3 runs, median	3 runs, median
Server platform	m6a.4xlarge	m6a.24xlarge
BIOS name and version	Amazon EC2 1.0 10/16/2017	Amazon EC2 1.0 10/16/2017
Operating system name and version/build number	Ubuntu 20.04 5.4.0-1045-aws	Ubuntu 20.04 5.4.0-1045-aws
Date of last OS updates/ patches applied	FP32,BS=1: 22/12/2021 FP32,BS=128: 21/12/2021 INT8,BS=1: 21/12/2021 INT8,BS=128: 21/12/2021	FP32,BS=1: 22/12/2021 FP32,BS=128: 22/12/2021 INT8,BS=1: 21/12/2021 INT8,BS=128: 21/12/2021

System configuration information	M6a (16 vCPU)	M6a (96 vCPU)
Processor		
Number of processors	1	2
Vendor and model	AMD EPYC 7R13 Processor	AMD EPYC 7R13 Processor
Core count (per processor)	32	32
Core frequency (GHz)	2.62	2.62
Stepping	1	1
Hyper-threading	Yes	Yes
Turbo	Yes	Yes
Number of vCPU per VM	16	96
Memory module(s)		
Total memory in system (GB)	64	384
NVMe memory present?	No	No
Total memory (DDR+NVMe RAM)	64	384
General HW		
Storage: NW or Direct Att / Instance	Direct Att / Instance	Direct Att / Instance
Local storage		
OS		
Number of drives	1	1
Drive size (GB)	30	30
Drive information (speed, interface, type)	Standard SSD (GP2)	Standard SSD (GP2)
Data drive		
Number of drives	1	1
Drive size (GB)	200	200
Drive information (speed, interface, type)	Standard SSD (GP2)	Standard SSD (GP2)
Network adapter		
Vendor and model	Amazon Elastic Network Adapter	Amazon Elastic Network Adapter
Number and type of ports	1x 12.5Gb	1x 37.5Gb

Table 7: Detailed information on the M6i instances we tested.

System configuration information	M6i (16 vCPU)	M6a (96 vCPU)
Tested by	Principled Technologies	Principled Technologies
Test date	FP32,BS=1: 22/12/2021 FP32,BS=128: 22/12/2021 INT8,BS=1: 21/12/2021 INT8,BS=128: 21/12/2021	FP32,BS=1: 22/12/2021 FP32,BS=128: 22/12/2021 INT8,BS=1: 21/12/2021 INT8,BS=128: 21/12/2021
CSP / Region	AWS EC2 – US East 1	AWS EC2 – US East 1
Workload & version	Model Zoo for Intel Architectures v2.4.0: resnet50v1_5	Model Zoo for Intel Architectures v2.4.0: resnet50v1_5
WL specific parameters	<i>See table 9: tuning profile table</i> FP32,BS=1: Profile E FP32,BS=128: Profile D INT8,BS=1: Profile F INT8,BS=128: Profile D	<i>See table 9: tuning profile table</i> FP32,BS=1: Profile E FP32,BS=128: Profile D INT8,BS=1: Profile F INT8,BS=128: Profile D
Iterations and result choice	3 runs, median	3 runs, median
Server platform	m6i.4xlarge	m6i.24xlarge
BIOS name and version	Amazon EC2 1.0 10/16/2017	Amazon EC2 1.0 10/16/2017
Operating system name and version/build number	Ubuntu 20.04 5.4.0-1045-aws	Ubuntu 20.04 5.4.0-1045-aws
Date of last OS updates/ patches applied	FP32,BS=1: 22/12/2021 FP32,BS=128: 22/12/2021 INT8,BS=1: 21/12/2021 INT8,BS=128: 21/12/2021	FP32,BS=1: 22/12/2021 FP32,BS=128: 22/12/2021 INT8,BS=1: 21/12/2021 INT8,BS=128: 21/12/2021
Processor		
Number of processors	1	2
Vendor and model	Intel Xeon 8375C	Intel Xeon 8375C
Core count (per processor)	32	32
Core frequency (GHz)	2.9	2.9
Stepping	6	6
Hyper-threading	Yes	Yes
Turbo	Yes	Yes
Number of vCPU per VM	16	96
Memory module(s)		
Total memory in system (GB)	64	384
NVMe memory present?	No	No
Total memory (DDR+NVMe RAM)	64	384
General HW		
Storage: NW or Direct Att / Instance	Direct Att / Instance	Direct Att / Instance

System configuration information	M6i (16 vCPU)	M6a (96 vCPU)
Local storage		
OS		
Number of drives	1	1
Drive size (GB)	30	30
Drive information (speed, interface, type)	Standard SSD (GP2)	Standard SSD (GP2)
Data drive		
Number of drives	1	1
Drive size (GB)	200	200
Drive information (speed, interface, type)	Standard SSD (GP2)	Standard SSD (GP2)
Network adapter		
Vendor and model	Amazon Elastic Network Adapter	Amazon Elastic Network Adapter
Number and type of ports	1x 12.5Gb	1x 37.5Gb

How we tested

This report builds on the methodology described below. We started with AWS instances with Intel or AMD processors and an Ubuntu operating system, installed Docker and Python, and built Docker containers with Model Zoo for Intel Architectures and prerequisites such as TensorFlow, tcmalloc, and more. We ran the benchmark in containers with settings we determined from a tuning process that explored the impact of various tunable parameters. For each configuration/host, we performed the test three times and selected the run with the median throughput (frames/s) as the characteristic run. We compared the median runs to those with similar geometry in either processor vendor (Intel vs. AMD) or processor generation.

Methodology for Intel processor instances used the unaltered Model Zoo code, while methodology for AMD processors involved altering the docker image build process to incorporate ZenDNN and use an AMD-optimized analog of the Intel-optimized default stack in Model Zoo for Intel Architectures. We accomplished this with a set of wrapper scripts to provide a common interface for both AMD and Intel testing. This code is available upon request by emailing info@principledtechnologies.com.

Instance types

For these tests, we used the following AWS instance types:

Table 8: The instances we used for testing

Instance**	Processor vendor	Processor model	HT/SMT	VCPUs	Memory (GiB)	OS drive* (GiB)	Data drive* (GiB)
m5n.4xlarge	Intel	Xeon Platinum 8259CL @ 2.5 GHz	Yes	16	64	30	200
m5n.24xlarge	Intel	Xeon Platinum 8259CL @ 2.5 GHz	Yes	96	384	30	200
m6i.4xlarge	Intel	Xeon Platinum 8375C @ 2.9 GHz	Yes	16	64	30	200
m6i.24xlarge	Intel	Xeon Platinum 8375C @ 2.9 GHz	Yes	96	384	30	200
m6a.4xlarge	AMD	EPYC 7R13 @ 2.65 GHz	Yes	16	64	30	200
m6a.24xlarge	AMD	EPYC 7R13 @ 2.65 GHz	Yes	96	384	30	200

*All drive volumes were standard GP2 EBS volumes.

**All instances used Ubuntu 20.04 AMI for base image.

Software stacks and versions

For these tests, we aimed to provide software stacks with identical framework and library versions where possible. However, we were unable to match versions exactly due to constraints of software compatibility. Table 9 defines the software versions used for the primary stack components for each test configuration. Note: These refer to the containers in which the benchmark runs, and not the host operating system of the AWS images.

Table 9: Software versions for primary stack components.

CPU vendor	Intel		AMD	
Model	Resnet50 v1.5		Resnet50 v1.5	
Precision	FP32	INT8	FP32	INT8
Operating system	Ubuntu 20.04	Ubuntu 20.04	Ubuntu 20.04	Ubuntu 20.04
Model Zoo for Intel Architectures	2.4.0	2.4.0	2.4.0*	2.4.0*
TensorFlow	2.4.0	2.4.0	2.5.0	2.5.0
Python	3.8.10	3.8.10	3.8.10	3.8.10
ZenDNN	N/A for Intel		3.1	3.1

*We made modifications to support AMD processors, and describe these changes in the appropriate sections below.

Provisioning AWS instances

1. Log into the AWS Console, and navigate to the EC2 dashboard.
2. In the left-hand menu, click Instances.
3. Click Launch Instances.
4. Under Step 1: Choose an Amazon Machine Image (AMI), locate Ubuntu Server 20.04 LTS (HVM) and SSD Volume Type.
5. Ensure 64-bit is selected.
6. Click Select.
7. Select one of the instance types (see the AWS Instance Types section of this document).
8. Click Next: Configure Instance Details.
9. Ensure the instance has a public IP address, and that HT/SMT is enabled under CPU options.
10. Click Next: Add Storage.
11. Set the Root volume's size to be 30GiB.
12. Click Add New Volume.
13. Leave everything default on the new volume except for its size.
14. Set the size to 200 GiB.
15. Click Next: Add Tags.
16. Add any tags required by your organization.
17. Click on Next: Configure Security Group.
18. Either create a new security group, or add the instance to an existing security group.
19. Ensure SSH port 22 is open in the security group from your local IP address.
20. Click Review and Launch.
21. On the review page, check that the instance settings match configuration specified above.
22. Click Launch.
23. Either create a new SSH keypair, or specify to use an existing keypair. Make sure you download the key files to a safe place, and install them in your SSH client's search path.
24. Click Launch Instance.
25. Watch the status of the EC2 instance. Once its status is healthy (running, 2/2 health checks passed), SSH into the instance using the appropriate key file.

Installing the prerequisite software packages

1. SSH into the AWS instance:

```
ssh -i AWS_INSTANCE_KEY_FILE ubuntu@AWS_INSTANCE_IP
```
2. Become root:

```
sudo su
```
3. Update APT cache:

```
apt-get update -y
```
4. Install utility packages:

```
apt-get install -y git wget htop curl tree jq ntp ntpdate openssh-server rsync
```
5. Configure GIT:

```
git config --global user.email "YOUR_EMAIL_ADDRESS"  
git config --global user.name "YOUR_NAME"
```
6. Install Python3:

```
apt-get install -y build-essential python3 python3-setuptools python3-dev python3-wheel \  
python3-pip python3-venv
```
7. Upgrade Python PIP:

```
pip install --upgrade pip
```

8. Install Docker prerequisites:

```
apt-get install -y aptitude expect ca-certificates gnupg lsb-release
```

9. Add Docker's official GPG key:

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o \
/usr/share/keyrings/docker-archive-keyring.gpg
```

10. Add the stable Docker repository to APT:

```
echo "deb [arch=$(dpkg --print-architecture) \
signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] \
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | \
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

11. Update APT:

```
cacheapt-get update -y
```

12. Install Docker Engine:

```
apt-get install docker-ce docker-ce-cli containerd.io
```

13. Verify docker is installed correctly:

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

14. Install Python packages useful for working with Docker:

```
pip install docker jsondiff docker-compose enum34
```

15. Install NMON:

```
apt-get install -y nmon
```

Preparing the data volume

1. SSH into the AWS instance:

```
ssh -i AWS_INSTANCE_KEY_FILE ubuntu@AWS_INSTANCE_IP
```

2. Become root:

```
sudo su
```

3. Create the persistent volume mount point:

```
mkdir /persistent
```

4. Create a partition on the persistent volume:

```
parted /dev/nvme1n1
mktable gpt
mkpart none ext2 2048kb 100%
print
quit
```

5. Format the persistent volume with EXT4:

```
mkfs -t ext4 /dev/nvme1n1p1
```

6. Mount the persistent volume:

```
mount -t ext4 /dev/nvme1n1p1 /persistent
```

7. Add the persistent volume to /etc/fstab:

```
echo -e "/dev/nvme1n1p1\t/persistent\ttext4\tdefaults 0 0" >> /etc/fstab
```

8. Create folder structure on persistent volume:

```
mkdir -p /persistent/datasets/resnetmkdir -p /persistent/models/resnet50v1_5
```

9. Follow external instructions to prepare the dataset:

<https://github.com/IntelAI/models/blob/master/datasets/imagenet/README.md>

10. Once complete, copy the dataset to persistent volume and delete the directory you created while getting the dataset:

```
mv ${IMAGENET_DIR}/* \ /persistent/models/resnet/
rm -rf ${IMAGENET_DIR}
```

11. Download the pretrained model for precision FP32:

```
cd /persistent/models/resnet50v1_5
wget https://zenodo.org/record/2535873/files/resnet50_v1.pb
```

12. Download the pretrained model for precision INT8:

```
cd /persistent/models/resnet50v1_5
wget
https://storage.googleapis.com/intel-optimized-tensorflow/models/v1_8/resnet50v1_5_int8_pretrained_model.pb
```

Note: The hyperlink above is one line.

Gathering the base image resources (M6a instances only)

For steps 4 and 7, please contact Principled Technologies at info@principledtechnologies.com for the script and Docker file. The ZenDNN installer script installs ZenDNN 3.1 and creates an environment file used when running the benchmark, and the ZenDNN 3.1 TensorFlow 2.5.0 Dockerfile defines the docker image with software stack including TensorFlow 2.5.0 and integrated ZenDNN 3.1.

1. On your local machine, open a terminal.
2. Create a directory structure:

```
mkdir -p /tmp/amd-base-image/build/tensorflow-v2.5.0/binaries
mkdir -p /tmp/amd-base-image/build-common/binaries
```

3. Download AMD AOCL 3.0 for Linux:

- a. In a web browser, navigate to <https://developer.amd.com/amd-aocl/archives/>.
- b. Scroll to the section AOCL 3.0 compiled with AOCC 3.0.
- c. Click `aocl-linux-aocc-3.0-6_1_amd64.deb`, and download it.
- d. Save the file as `/tmp/amd-base-image/build-common/binaries/aocl-linux-aocc-3.0-6_1_amd64.deb`.

4. Create the ZenDNN installer script:

- a. Contact info@principledtechnologies.com to request a copy of the ZenDNN installer script.
- b. Create a new file with the nano editor:

```
nano /tmp/amd-base-image/build-common/binaries/install-zendnn.sh
```

- c. Copy and paste the content from the script you received in step 1.
- d. To save the file, press CTRL+O.
- e. To exit nano, press CTRL+X.

5. Download AMD AOCC compiler 3.1:

- a. In a web browser, navigate to <https://developer.amd.com/aocc-archive-section/>.
- b. Scroll down to the section AOCC v3.1.
- c. Click `aocc-compiler-3.1.0_1_amd64.deb`, and download it.
- d. Save the file as `/tmp/amd-base-image/build/tensorflow-v2.5.0/binaries/aocc-compiler-3.1.0_1_amd64.deb`.

6. Download AMD ZenDNN 3.1:

- a. In a web browser, navigate to <https://developer.amd.com/zendnn/zendnn-archives/>.
- b. Scroll down to the section ZenDNN 3.1.
- c. Click `TF_v2.5_ZenDNN_v3.1.zip`, and download it.
- d. Save the file as `/tmp/amd-base-image/build/tensorflow-v2.5.0/binaries/TF_v2.5_ZenDNN_v3.1.zip`.

7. Create the TensorFlow v2.5.0 Docker file.

- a. Contact info@principledtechnologies.com to request a copy of the ZenDNN installer script.
- b. Create a new file with the nano editor:

```
nano /tmp/amd-base-image/build/tensorflow-v2.5.0/Dockerfile
```

- c. Copy and paste the content from the dockerfile you received in step 1.
- d. To save the file, press CTRL+O.
- e. To exit nano, press CTRL+X.

Preparing the AMD TensorFlow 2.5.0 base image (M6a instances only)

1. On your local machine, open a terminal.
2. Copy the build files to the AWS instance:

```
cd /tmp/amd-base-image/build/  
rsync -rav -e "ssh -i AWS_INSTANCE_KEY_FILE" \  
./tensorflow-v2.5.0/ ubuntu@AWS_INSTANCE_IP:~/amd-tensorflow-v2.5.0/  
rsync -rav -e "ssh -i AWS_INSTANCE_KEY_FILE" \  
../build-common/ ubuntu@AWS_INSTANCE_IP:~/amd-tensorflow-v2.5.0/
```

3. SSH into the AWS instance:

```
ssh -i AWS_INSTANCE_KEY_FILE ubuntu@AWS_INSTANCE_IP
```

4. Become root :

```
sudo su
```

5. Change to the build directory:

```
cd /home/ubuntu/amd-tensorflow-v2.5.0/
```

6. Build and tag the docker image:

```
docker build -f Dockerfile -t amd/tensorflow:v2.5.0
```

Installing Model Zoo for Intel Architectures

1. SSH into the AWS instance:

```
ssh -i AWS_INSTANCE_KEY_FILE ubuntu@AWS_INSTANCE_IP
```

2. Become root:

```
sudo su
```

3. Create the install directory:

```
mkdir -p /opt/imz-tools/intel-model-zoo
```

4. Change to the install directory's parent directory:

```
cd /opt/imz-tools
```

5. Clone the Model Zoo git repository:

```
git clone https://github.com/IntelAI/models ./intel-model-zoo
```

6. Check out version 2.4.0:

```
git checkout v2.4.0
```

7. Create a base branch:

```
git checkout -b v2.4.0-base
```

Applying a patch file for AMD support (M6a instances only)

For step 1, please contact Principled Technologies at info@principledtechnologies.com for the AMD patch. The patch creates AMD-suitable docker image slice definitions and provides a small patch to the benchmark CPU binding logic. Note that the primary change accomplished with alternate slice definitions is the replacement of the base image; aside from some naming-related differences and the small patch to CPU binding logic, the software stack above the base image is identical.

1. Contact info@principledtechnologies.com to request a copy of the AMD/ZenDNN patch for Model Zoo for Intel Architectures.
2. Save the patch file as `~/amd.patch`.
3. Copy the AMD patch files from your local machine to the AWS instance:

```
rsync -e "ssh -I AWS_INSTANCE_KEY_FILE" -av ~/amd.patch ubuntu@AWS_INSTANCE_IP:~/amd.patch
```

4. SSH into the AWS instance:

```
ssh -i AWS_INSTANCE_KEY_FILE ubuntu@AWS_INSTANCE_IP
```

5. Become root:

```
sudo su
```

6. Change to the install directory:

```
cd /opt/imz-tools/intel-model-zoo
```

7. Checkout the base branch:

```
git checkout v2.4.0-base
```

8. Create a new branch to hold patched version of code:

```
git checkout -b v2.4.0-amd
```

9. Apply patch file to the newly created branch:

```
git apply /home/ubuntu/amd.patch
```

10. Commit changes to the branch:

```
git add . && git commit -m "Applied AMD patch files."
```

11. Remove the patch file:

```
rm -rf /home/ubuntu/amd.patch
```

Creating the model builder tools Docker image

1. SSH into the AWS instance:

```
ssh -i AWS_INSTANCE_KEY_FILE ubuntu@AWS_INSTANCE_IP
```

2. Become root:

```
sudo su
```

3. Change to the tools directory:

```
cd /opt/imz-tools/intel-model-zoo/tools
```

4. List the models (this will trigger tools image to be built):

```
bash ./scripts/model-builder -f tensorflow images
```

Creating the benchmark image

1. SSH into the AWS instance:

```
ssh -i AWS_INSTANCE_KEY_FILE ubuntu@AWS_INSTANCE_IP
```

2. Become root:

```
sudo su
```

3. Change to the tools directory:

```
cd /opt/imz-tools/intel-model-zoo/tools
```

4. Invoke the image builder using the MODEL_NAME from Table 10:

```
bash ./scripts/model-builder -f tensorflow make MODEL_NAME
```

Table 10: Model names for the image builder.

CPU Vendor	Precision	MODEL_NAME
AMD	FP32	resnet50v1-5-fp32-inference
AMD	INT8	resnet50v1-5-int832-inference
Intel	FP32	resnet50v1-5-fp32-inference
Intel	INT8	resnet50v1-5-int832-inference

Starting NMON performance data collection

- SSH into the AWS instance:


```
ssh -i AWS_INSTANCE_KEY_FILE ubuntu@AWS_INSTANCE_IP
```
- Become root:


```
sudo su
```
- Start NMON:


```
nmon -F /tmp/output.nmon -s 1 -c 36000 -J -t -p > /tmp/nmon.pid
```

Invoking the benchmark

- SSH into the AWS instance:


```
ssh -i AWS_INSTANCE_KEY_FILE ubuntu@AWS_INSTANCE_IP
```
- Become root:


```
sudo su
```
- Using the substitutions from Table 11, run the benchmark in a Docker container.

Note: Substitutions not listed in the table are tunable parameters, and vary by cpu/model/precision/batch-size. See Appendix 1: Tuning profiles for more information.

Table 11: List of substitutions.

CPU vendor	Precision	MODEL_NAME MODEL_MOUNT DS_MOUNT DOCKER_IMAGE IN_GRAPH DATA_LOCATION EXTRA_COMMAND
Intel	fp32	resnet50v1_5 /persistent/models/resnet50v1_5 /persistent/datasets/resnet intel-model-zoo:latest-image-recognition-resnet50-fp32-inference /models/resnet50_v1.pb /datasets <no extra command>

CPU vendor	Precision	MODEL_NAME MODEL_MOUNT DS_MOUNT DOCKER_IMAGE IN_GRAPH DATA_LOCATION EXTRA_COMMAND
Intel	int8	resnet50v1_5 /persistent/models/resnet50v1_5 /persistent/datasets/resnet intel-model-zoo:latest-image-recognition-resnet50-int8-inference /models/resnet50v1_5_int8_pretrained_model.pb /datasets <no extra command>
AMD	fp32	resnet50v1_5 /persistent/models/resnet50v1_5 /persistent/datasets/resnet amd-model-zoo:latest-amd-image-recognition-resnet50v1-5-fp32-inference /models/resnet50v1_5_int8_pretrained_model.pb /datasets ./ZenDNNrc && \
AMD	int8	resnet50v1_5 /persistent/models/resnet50v1_5 /persistent/datasets/resnet amd-model-zoo:latest-amd-image-recognition-resnet50v1-5-int8-inference /models/resnet50v1_5_int8_pretrained_model.pb /datasets ./ZenDNNrc && \

```

docker run \
--rm \
--privileged \
-v /opt/imz-tools/results:/output \
-v MODEL_MOUNT:/models \
-v DS_MOUNT:/datasets \
-v /opt/imz-tools/intel-model-zoo:/intel-model-zoo \
-w /intel-model-zoo \
DOCKER_IMAGE \
/bin/bash -c '\
export DEBIAN_FRONTEND=noninteractive && \
cd /intel-model-zoo && \
apt-get update -y && \
apt-get install -y numactl util-linux && \
apt-get install -y google-perftools && \
python3.7 -m pip install pandas ptutils pyyaml &&\
export LD_PRELOAD="/usr/lib/x86_64-linux-gnu/libtcmalloc.so.4.3.0${LD_PRELOAD:+:${LD_

```

```

PRELOAD}}" && \
export KMP_AFFINITY="verbose,granularity=fine,compact" && \
EXTRA_COMMAND
python3 /intel-model-zoo/benchmarks/launch_benchmark.py \
--output-dir=/output/NAME_OF_TEST_RUN \
--model-name MODEL_NAME \
--precision PRECISION \
--mode inference \
--framework tensorflow \
--benchmark-only \
--batch-size BATCH_SIZE \
--in-graph IN_GRAPH \
--data-location DATA_LOCATION \
--numa-cores-per-instance NUMA_CORES_PER_INSTANCE \
--num-cores NUM_CORES \
--num-intra-threads NUM_INTRA_THREADS \
--num-inter-threads NUM_INTER_THREADS \
--disable-tcmalloc=False \
--num_omp_threads= NUM_OMP_THREADS \
--warmup-steps=WARMUP_STEPS \
--steps=STEPS \

```

Collecting NMON performance data

1. SSH into the AWS instance:

```
ssh -i AWS_INSTANCE_KEY_FILE ubuntu@AWS_INSTANCE_IP
```

2. Become root:

```
sudo su
```

3. Stop NMON

```
kill -s USR2 'cat /tmp/nmon.pid'
```

4. Change output file ownership:

```
chown ubuntu:ubuntu /tmp/output.nmon
```

5. Open a terminal on your local machine.

6. Copy the output file to your local machine.

```
rsync -e "ssh -I AWS_INSTANCE_KEY_FILE" -av ubuntu@AWS_INSTANCE_IP:/tmp/output.nmon ./output.nmon
```

Tuning profiles

Tests for each unique combination of CPU, Precision, batch-size used tuning parameters defined in Table 12.

Table 12: Tuning profiles for testing.

Profile	A	D	E	F	G	H	I	J
NUM_INSTANCES	1	1	1	1	1	1	1	2
NUM_INTRA_THREADS	4	8	2	4	4	4	4	4
NUM_INTER_THREADS	2	2	2	1	4	2	2	2
NUM_OMP_THREADS	4	4	4	4	4	4	4	4
NUMA_CORES_PER_INSTANCE	4	4	4	4	4	2	8	8
NUM_CORES	4	4	4	4	4	2	8	8

Table 13 identifies the tuning profile used for each benchmark run. Profiles were selected by running the benchmarks with each profile and determining which profile provided the maximum throughput.

Table 13: Tuning profile for each benchmark run.

Instance type	vCPUs	Precision	Batch size	Profile selected
m5n.4xlarge	16	fp32	1	E
m5n.24xlarge	96	fp32	1	E
m5n.4xlarge	16	fp32	128	D
m5n.24xlarge	96	fp32	128	D
m5n.4xlarge	16	int8	1	F
m5n.24xlarge	96	int8	1	F
m5n.4xlarge	16	int8	128	D
m5n.24xlarge	96	int8	128	D
m6i.4xlarge	16	fp32	1	E
m6i.24xlarge	96	fp32	1	E
m6i.4xlarge	16	fp32	128	D
m6i.24xlarge	96	fp32	128	D
m6i.4xlarge	16	int8	1	F
m6i.24xlarge	96	int8	1	F
m6i.4xlarge	16	int8	128	D
m6i.24xlarge	96	int8	128	D
m6a.4xlarge	16	fp32	1	E
m6a_24xlarge	96	fp32	1	E
m6a.4xlarge	16	fp32	128	F
m6a.24xlarge	96	fp32	128	D
m6a.4xlarge	16	int8	1	E
m6a.24xlarge	96	int8	1	E
m6a.4xlarge	16	int8	128	D
m6a.24xlarge	96	int8	128	D

Read the report at <https://facts.pt/DNLGDAd> ▶

This project was commissioned by Intel.



Facts matter.®

Principled Technologies is a registered trademark of Principled Technologies, Inc. All other product names are the trademarks of their respective owners.

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.