Use ROCm™ on Radeon™ GPUs Documentation

Advanced Micro Devices, Inc.

Jul 12, 2024

CONTENTS

Turn your desktop into a Machine Learning platform with the latest high-end AMD Radeon™ 7000 series GPUs

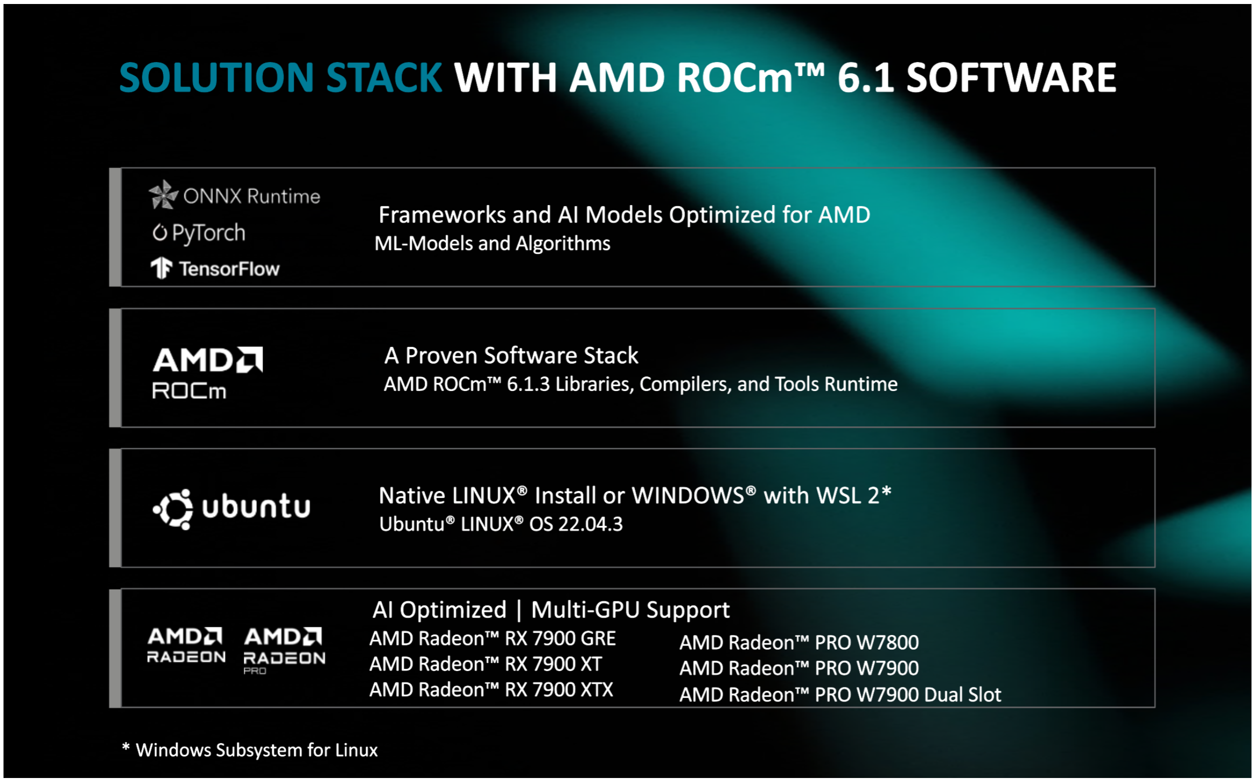
AMD has expanded support for Machine Learning Development on RDNA™ 3 GPUs with Radeon™ Software for Linux 24.10.3 with ROCm™ 6.1.3!

Researchers and developers working with Machine Learning (ML) models and algorithms using PyTorch, ONNX Runtime, or TensorFlow can now also use ROCm 6.1.3 on Linux® to tap into the parallel computing power of the latest high-end AMD Radeon 7000 series desktop GPUs, and based on AMD RDNA 3 GPU architecture.

A client solution built on powerful high-end AMD GPUs enables a local, private, and cost-effective workflow to develop ROCm and train Machine Learning for users who were solely reliant on cloud-based solutions.

More ML performance for your desktop

- With today's models easily exceeding the capabilities of standard hardware and software not designed for AI, ML engineers are looking for cost-effective solutions to develop and train their ML-powered applications. Due to the availability of significantly large GPU memory sizes of 24GB or 48GB, utilization of a local PC or workstation equipped with the latest high-end AMD Radeon 7000 series GPU offers a robust/potent yet economical option to meet these expanding ML workflow challenges.
- Latest high-end AMD Radeon 7000 series GPUs are built on the RDNA 3 GPU architecture,
  - featuring more than 2x higher AI performance per Compute Unit (CU) compared to the previous generation
  - now comes with up to 192 AI accelerators
  - offers up to 24GB or 48GB of GPU memory to handle large ML models



SOLUTION STACK WITH AMD ROCm™ 6.1 SOFTWARE

ONNX Runtime
PyTorch
TensorFlow

Frameworks and AI Models Optimized for AMD
ML-Models and Algorithms

AMD ROCm

A Proven Software Stack
AMD ROCm™ 6.1.3 Libraries, Compilers, and Tools Runtime

ubuntu

Native LINUX® Install or WINDOWS® with WSL 2*
Ubuntu® LINUX® OS 22.04.3

AMD RADEON AMD RADEON PRO

AI Optimized | Multi-GPU Support

AMD Radeon™ RX 7900 GRE
AMD Radeon™ RX 7900 XT
AMD Radeon™ RX 7900 XTX

AMD Radeon™ PRO W7800
AMD Radeon™ PRO W7900
AMD Radeon™ PRO W7900 Dual Slot

* Windows Subsystem for Linux

Note: Based on AMD internal measurements, November 2022, comparing the Radeon RX 7900 XTX at 2.505 GHz boost clock with 96 CUs issuing 2X the Bfloat16 math operations per clock vs. the Radeon RX 6900 XT GPU at 2.25 GHz boost clock and 80 CUs issue 1X the Bfloat16 math operations per clock. Results may vary. RX-821.

Migrate your application from the desktop to the datacenter

- ROCm is the open-source software stack for Graphics Processing Unit (GPU) programming. ROCm spans several domains: General-Purpose computing on GPUs (GPGPU), High Performance Computing (HPC) and heterogeneous computing.
- The latest AMD ROCm 6.1.3 software stack for GPU programming unlocks the massively parallel compute power of these RDNA 3 GPUs for use with various ML frameworks. The same software stack also supports AMD CDNA™ GPU architecture, so developers can migrate applications from their preferred framework into the datacenter.

Freedom to customize

ROCm is primarily Open-Source Software (OSS) that allows developers the freedom to customize and tailor their GPU software for their own needs while collaborating with a community of other developers, and helping each other find solutions in an agile, flexible, rapid and secure manner. AMD ROCm allows users to maximize their GPU hardware investment. ROCm is designed to help develop, test and deploy GPU accelerated HPC, AI, scientific computing, CAD, and other applications in a free, open-source, integrated and secure software ecosystem.

Improved interoperability

- Support for PyTorch, one of the leading ML frameworks.
- Support for ONNX Runtime to perform inference on a wider range of source data, including INT8 with MIGraphX.
- Support for TensorFlow.

Note: Visit AMD ROCm Documentation for the latest on ROCm.

For the latest driver installation packages, visit Linux Drivers for Radeon Software.

CHAPTER

ONE

PREREQUISITES TO USE ROCM ON RADEON DESKTOP GPUS FOR MACHINE LEARNING DEVELOPMENT

Before starting with the installation, ensure that your system meets the necessary requirements such as supported hardware, a compatible operating system, and the recommended system configuration to ensure optimal performance and functionality.

See *Compatibility matrices* for more information.

1.1 Supported hardware

1.1.1 Supported graphics processing units

To successfully install ROCm™ for machine learning development, ensure that your system is operating on a Radeon™ Desktop GPU listed in the *Compatibility matrices* section.

1.1.2 Recommended memory

The recommended memory to use ROCm on Radeon. These specifications are required for complex AI/ML workloads:

- 64GB Main Memory
- 24GB GPU Video Memory

  Note AMD recommends having the same amount of system memory as video memory, as a minimum.

Minimum recommendations

Minimum memory requirements to use ROCm on Radeon. Note that low system memory may cause issues running inference models.

- 16GB Main Memory
- 8GB GPU Video Memory

  Important! These are guidelines only. Note that minimum memory required will vary depending on workload.

1.2 Supported operating systems

Ensure that your operating system is up-to-date to successfully install ROCm for machine learning development.

Refer to *Compatibility matrices* for up-to-date operating system compatibility.

1.2.1 Update Ubuntu® operating system

Use the following commands to bring your OS up-to-date:

```
sudo apt-get update
sudo apt-get dist-upgrade
```

1.3 Recommended system configuration

This section guides users on how to optimize system configuration for ROCm™ usage, ensuring smooth and performant ROCm operation.

1.3.1 Disable iGPU

The iGPU is non-essential for AI and ML workloads and not officially supported. Disable iGPU in SBIOS before proceeding to avoid unknown issues.

Alternatively, use environment variables to select the target GPU.

Here are examples to disable iGPU on some AMD motherboards:

Gigabyte™ X670 AORUS ELITE AX

1. Enter BIOS

   Path: *Advanced → AMD CBS → NBIO Common Options → GFX Configuration → iGPU Configuration*

2. Set iGPU to `Disabled`

ASUS Prime X670-P WIFI

1. Enter BIOS

   Path: *Advanced → NB Configuration → Integrated Graphics*

2. Set to `Disabled`

   NOTE: This step only applies to AMD motherboards, no action is required for non-AMD motherboards.

   There are no minimum motherboard hardware requirements.

Alternative option: Use environment variables to select target GPU

An alternative option to disabling the iGPU is to use environment variable to select the GPU.

See GPU Isolation Techniques to specify the device indices you would like to expose to your application.

CHAPTER

TWO

HOW TO GUIDES

These guides walk you through the various installation processes required to pair ROCm™ with the latest high-end AMD Radeon™ 7000 series desktop GPUs.

Linux

WSL

Linux How to guide

WSL How to guide

2.1 Linux How to guide - Use ROCm on Radeon GPUs

This guide walks you through the various installation processes required to pair ROCm™ with the latest high-end AMD Radeon™ 7000 series desktop GPUs, and get started on a fully-functional environment for AI and ML development.

2.1.1 Install Radeon software for Linux with ROCm

The ROCm™ Software Stack and other Radeon™ software for Linux components are installed using the `amdgpu-install` script to assist you in the installation of a coherent set of stack components.

- Simplifies the installation of the AMDGPU stack by encapsulating the distribution specific package installation logic and by using command line options that allows you to specify the:
  - Usecase of the AMDGPU stack to be installed (Graphics or Workstation)
  - Combination of components (Pro stack, or user selection)
- Performs post-install checks to verify whether the installation was performed successfully.
- Installs the uninstallation script to allow you to remove the whole AMDGPU stack from the system by using a single command.

The script is provided by the installer package. See *Compatibility matrices* for support information.

Install AMD unified driver package repositories and installer script

Download and install the `amdgpu-install` script on the system.

Enter the following commands to install the installer script for the latest compatible Ubuntu® version:

```
sudo apt update
wget https://repo.radeon.com/amdgpu-install/6.1.3/ubuntu/jammy/amdgpu-install_6.1.60103-1_all.deb
sudo apt install ./amdgpu-install_6.1.60103-1_all.deb
```

Install AMD unified kernel-mode GPU driver, ROCm, and graphics

After the Unified Driver Deb Package repositories are installed, run the installer script with appropriate `--usecase` parameters to install the driver components.

AMD recommends installing the Graphics usecase by default. Only consider the alternative install option if you have an applicable Workstation usecase scenario.

Enter the following command to display a list of available usecases:

```
sudo amdgpu-install --list-usecase
```

Option A: Graphics usecase

AMD recommends installing the Graphics usecase by default.

1. Run the following command to install open source graphics and ROCm.

   ```
   amdgpu-install -y --usecase=graphics,rocm
   ```

   Watch for output warning or errors indicating an unsuccessful driver installation.

   Note: The `-y` option installs non-interactively. This step may take several minutes, depending on internet connection and system speed.

2. Reboot the system.

   ```
   sudo reboot
   ```

   See Using the amdgpu-install script for more information.

Next, *set Groups permissions*.

Option B: Workstation usecase

Only consider the alternative install option if you have an applicable Workstation usecase scenario.

1. Run the following command to install workstation graphics and ROCm.

   ```
   amdgpu-install -y --usecase=workstation,rocm
   ```

   Note: The -y option installs non-interactively. This step may take several minutes, depending on internet connection and system speed.

2. Reboot the system.

```
sudo reboot
```

See Using the amdgpu-install script for more information.

Next, *set Groups permissions*.

Set Groups permissions

Once the driver is installed, add any current user to the render and video groups to access GPU resources.

Reboot in order for group changes to take effect.

Add user to render and video groups

1. Enter the following command to check groups in the system:

```
groups
```

2. Add user to the render and video group using the command:

```
sudo usermod -a -G render,video $LOGNAME
```

3. Reboot the system.

```
sudo reboot
```

See Setting Permissions for Groups for more information.

Post-install verification checks

Run these post-installation checks to verify that the installation is complete:

1. Verify that the current user is added to the render and video groups.

```
groups
```

Expected result:

```
<username> adm cdrom sudo dip video plugdev render lpadmin lxd sambashare
```

`<username>` indicates the current user, and this result will vary in your environment.

2. Check if *amdgpu* kernel driver is installed.

```
dkms status
```

Expected result:

```
amdgpu/x.x.x-xxxxxxx.xx.xx, x.x.x-xx-generic, x86_64: installed
```

3. Check if the GPU is listed as an agent.

```
rocminfo
```

Expected result:

```
[...]
*******
Agent 2
*******
  Name:                    gfx1100
  Uuid:                    GPU-5ecee39292e80c37
  Marketing Name:          Radeon RX 7900 XTX
  Vendor Name:             AMD
  [...]
[...]
```

4. Check if the GPU is listed.

```
clinfo
```

Expected result:

```
[...]
  Platform Name:                 AMD Accelerated Parallel Processing
Number of devices:               1
  Device Type:                   CL_DEVICE_TYPE_GPU
  Vendor ID:                     1002h
  Board name:                    Radeon RX 7900 XTX
[...]
```

See Installing the all open usecase for additional troubleshooting tips.

Advanced install methods

For advanced install methods, such as Multi-Version and Package Manager, refer to AMD GPU Install Script.

Uninstall ROCm

Run the following command to uninstall the ROCm software stack and other Radeon software for Linux components:

```
sudo amdgpu-uninstall
```

Upgrade to newer versions of Radeon software for Linux

The recommended method to upgrade is to uninstall, followed by an install.

> Radeon Software for Linux does not support in-place upgrades.

2.1.2 Install PyTorch for ROCm

Refer to this section for the recommended PyTorch via PIP installation method, as well as Docker-based installation.

Option A: PyTorch via PIP installation

AMD recommends the PIP install method to create a PyTorch environment when working with ROCm™ for machine learning development.

> Check Pytorch.org for latest PIP install instructions and availability. See *Compatibility matrices* for support information.
>
> **Note** The latest version of Python module numpy v2.0 is incompatible with the torch wheels for this version. Downgrade to an older version is required. Example: pip3 install numpy==1.26.4

Install PyTorch via PIP

1. Enter the following command to unpack and begin set up.

```
sudo apt install python3-pip -y
```

2. Enter this command to update the pip wheel.

```
pip3 install --upgrade pip wheel
```

3. Enter this command to install Torch and Torchvision for ROCm AMD GPU support.

```
wget https://repo.radeon.com/rocm/manylinux/rocm-rel-6.1.3/torch-2.1.2%2Brocm6.1.
↪3-cp310-cp310-linux_x86_64.whl
wget https://repo.radeon.com/rocm/manylinux/rocm-rel-6.1.3/torchvision-0.16.1
↪%2Brocm6.1.3-cp310-cp310-linux_x86_64.whl
wget https://repo.radeon.com/rocm/manylinux/rocm-rel-6.1.3/pytorch_triton_rocm-2.
↪1.0%2Brocm6.1.3.4d510c3a44-cp310-cp310-linux_x86_64.whl
pip3 uninstall torch torchvision pytorch-triton-rocm numpy
pip3 install torch-2.1.2+rocm6.1.3-cp310-cp310-linux_x86_64.whl torchvision-0.16.
↪1+rocm6.1.3-cp310-cp310-linux_x86_64.whl pytorch_triton_rocm-2.1.0+rocm6.1.3.
↪4d510c3a44-cp310-cp310-linux_x86_64.whl numpy==1.26.4
```

> This may take several minutes.
>
> **Important!** AMD recommends proceeding with ROCm WHLs available at repo.radeon.com. The ROCm WHLs available at PyTorch.org are not tested extensively by AMD as the WHLs change regularly when the nightly builds are updated.

Next, *verify your PyTorch installation*.

Option B: Docker installation

Using Docker provides portability, and access to a prebuilt Docker container that has been rigorously tested within AMD. Docker also cuts down compilation time, and should perform as expected without installation issues.

Prerequisites to install PyTorch using Docker

Docker for Ubuntu® must be installed.

To install Docker for Ubuntu, enter the following command:

```
sudo apt install docker.io
```

Use Docker image with pre-installed PyTorch

Follow these steps for installing using a Docker image.

1. Enter the following command to pull the public PyTorch Docker image.

```
sudo docker pull rocm/pytorch:rocm6.1.3_ubuntu22.04_py3.10_pytorch_release-2.1.2
```

   Optional: You can also download a specific and supported configuration with different user-space ROCm versions, PyTorch versions, and supported operating systems.

   Refer to hub.docker.com/r/rocm/pytorch to download the PyTorch Docker image.

2. Start a Docker container using the downloaded image.

```
sudo docker run -it \
  --cap-add=SYS_PTRACE \
  --security-opt seccomp=unconfined \
  --device=/dev/kfd \
  --device=/dev/dri \
  --group-add video \
  --ipc=host \
  --shm-size 8G \
  rocm/pytorch:rocm6.1.3_ubuntu22.04_py3.10_pytorch_release-2.1.2
```

   This will automatically download the image if it does not exist on the host. You can also pass the `-v` argument to mount any data directories from the host onto the container.

Next, *verify the PyTorch installation*.

See PyTorch Installation for ROCm for more information.

Verify PyTorch installation

Confirm if PyTorch is correctly installed.

1. Verify if Pytorch is installed and detecting the GPU compute device.

```
python3 -c 'import torch' 2> /dev/null && echo 'Success' || echo 'Failure'
```

   Expected result:

```
Success
```

2. Enter command to test if the GPU is available.

```
python3 -c 'import torch; print(torch.cuda.is_available())'
```

Expected result:

```
True
```

3. Enter command to display installed GPU device name.

```
python3 -c "import torch; print(f'device name [0]:', torch.cuda.get_device_
↪name(0))"
```

Expected result: Example: *device name [0]: Radeon RX 7900 XTX*

```
device name [0]: <Supported AMD GPU>
```

4. Enter command to display component information within the current PyTorch environment.

```
python3 -m torch.utils.collect_env
```

Expected result:

```
PyTorch version

ROCM used to build PyTorch

OS

Is CUDA available

GPU model and configuration

HIP runtime version

MIOpen runtime version
```

Environment set-up is complete, and the system is ready for use with PyTorch to work with machine learning models, and algorithms.

2.1.3 Install ONNX Runtime for Radeon GPUs

Refer to this section to install ONNX via the PIP installation method.

Overview

Ensure that the following prerequisite installations are successful before proceeding to install ONNX Runtime for use with ROCm™ on Radeon™ GPUs.

Prerequisites

- Radeon Software for Linux (with ROCm) is installed.
- *MIGraphX* is installed. This enables ONNX Runtime to build the correct MIGraphX Execution Provider (EP).
- The half library is installed. See *Verify if MIGraphX is installed with the half library*.
- If the prerequisite installations are successful, proceed to *install ONNX Runtime*.

  NOTE Unless adding custom features, use the pre-built Python wheel files provided in the PIP installation method.

Verify if MIGraphX is installed with the half library

```
$ dpkg -l | grep migraphx
$ dpkg -l | grep half
```

Expected result:

```
root@aus-navi3x-02:/workspace/AMDMIGraphX# dpkg -l | grep migraphx
ii  migraphx                      2.9.0                             amd64        AMD
 ↪'s graph optimizer
ii  migraphx-dev                  2.9.0                             amd64        AMD
 ↪'s graph optimizer
ii  migraphx-tests                2.9.0                             amd64        AMD
 ↪'s graph opt


$ dpkg -l | grep half
ii  half                          1.12.0.60000-91~20.04             amd64        HALF-
 ↪PRECISION FLOATING POINT LIBRARY
```

NOTE Versions may vary between ROCm builds and installed versions of MIGraphX, but the desired result is the same.

Enter the following install command if the half library is not installed with MIGraphX:

```
$ dpkg -l | grep migraphx
$ dpkg -l | grep half
```

NOTE The half library should come packaged with MIGraphX. If not, it can be installed with the following command:

```
$ sudo apt install half
```

Install ONNX Runtime

> **Important!**
>
> - Use the provided pre-built Python wheel files from the PIP installation method, unless adding custom features.
> - The wheel file contains the MIGraphX and ROCm Execution Providers (EP). Refer to *Install MIGraphX for ONNX RT* for more information.
> - Refer to ONNX Runtime Documentation for additional information on ONNX Runtime topics.
> - See ONNX Runtime Tutorials to try out real applications and tutorials on how to get started.

Option A: PIP install (Recommended)

Option A: ONNX Runtime install via PIP installation method (Recommended)

AMD recommends the PIP install method to create an ONNX Runtime environment when working with ROCm for machine learning development.

> **Note** The latest version of Python module numpy v2.0 is incompatible with the ONNX Runtime wheels for this version. Downgrade to an older version is required. Example: pip3 install numpy==1.26.4

To install via PIP,

Enter this command to download and install the ONNX Runtime wheel.

```
pip3 uninstall onnxruntime-rocm numpy
pip3 install https://repo.radeon.com/rocm/manylinux/rocm-rel-6.1.3/onnxruntime_rocm-1.
↪17.0-cp310-cp310-linux_x86_64.whl numpy==1.26.4
```

Next, *verify the ONNX Runtime installation*.

Option B: Build from source (Advanced)

Option B: Build from source for your environment, followed by local wheel file installation (Advanced)

Use this method for advanced customization usecases. This requires the user install the desired ROCm and MIGraphX versions, and creation of softlink prior to starting the build.

> **NOTE** The build time typically takes ~45 minutes.

Prerequisites to build ONNX from source

- Radeon Software for Linux (with ROCm) is installed
- *MigraphX* is installed
- Softlink is created

  To create a softlink for `/opt/rocm`, enter the following command:

  ```
  language = bash
  linenumbers = true
  ln -s /opt/rocm* /opt/rocm
  ```

To build from source,

1. Clone the ONNX Runtime repository into the root directory.

```
cd /
git clone https://github.com/microsoft/onnxruntime.git
```

2. Git clone AMDMIGraphX into the home folder.

```
cd ~
git clone https://github.com/ROCm/AMDMIGraphX.git
```

3. Create a docker image for MIGraphX.

   Note

   - Refer to AMDMIGraphX Github for up-to-date ONNX Runtime and MIGraphX dependencies.
   - MIGraphX can still be built or installed from apt.
   - For MIGraphX package builds via **RBuild**, refer to these MIGraphx Github instructions to build within the docker container environment.
   - For MIGraphX package builds via **CMake**, refer to these MIGraphx Github instructions to build within the docker container environment.

- Use the groups command to ensure that the user is part of the video, render, and docker groups in Linux to run the docker container.

```
groups
tthemist@aus-navi3x-02 ~/groups
tthemist sudo video render docker
```

- Run the following for a simple MIGraphX apt install:

```
cd AMDMIGraphX
docker build -t migraphx .
docker run --device='/dev/kfd' --device='/dev/dri' -v=`pwd`:/code/AMDMIGraphX␣
↪-v /onnxruntime:/onnxruntime -w /code/AMDMIGraphX --group-add video -it␣
↪migraphx
apt install migraphx migraphx-dev half
```

4. Run rocm-smi to ensure that ROCm is installed and detects the supported GPU(s).

```
$ rocm-smi
```

Expected result:

```
======================================= ROCm System Management Interface␣
↪=======================================
================================================= Concise Info␣
↪=================================================
Device  [Model : Revision]    Temp    Power  Partitions      SCLK     MCLK    Fan ␣
↪    Perf  PwrCap  VRAM%  GPU%
        Name (20 chars)       (Edge)  (Avg)  (Mem, Compute)
=================================================================================================
0       [0x0e0d : 0x00]       32.0°C  73.0W  N/A, N/A        1526Mhz  96Mhz  31.76
↪%  auto  241.0W    0%   50%
        0x7448
=================================================================================================
```

(continues on next page)

(continued from previous page)

```
============================================ End of ROCm SMI Log ============================================
```

5. Configure Git to treat all directories as safe to use and run the build script.

```
cd AMDMIGraphX
git config --global --add safe.directory "*"
tools/build_and_test_onnxrt.sh
```

   This builds ONNX Runtime and adds ROCm and MIGraphX EP support to the ONNX Runtime interface and requires multiple external repo pieces be checked out automatically prior to the build.

6. Install ONNX Runtime once MIGraphX is built.

```
$ pip3 install /onnxruntime/build/Linux/Release/dist/*.whl
```

Next, *verify your ONNX Runtime installation*.

Verify ONNX Runtime installation

Confirm if ONNX Runtime is correctly installed.

```
$ python3
$ import onnxruntime as ort
  ort.get_available_providers()
```

Expected result: The following EPs are displayed.

```
>>> ort.get_available_providers()
['MIGraphXExecutionProvider', 'ROCMExecutionProvider', 'CPUExecutionProvider']
```

Environment set-up is complete, and the system is ready for use with ONNX Runtime to work with machine learning models, and algorithms.

2.1.4 Install TensorFlow for ROCm

TensorFlow is an open-source library for solving machine-learning, deep-learning, and artificial-intelligence problems. It can be used to solve many problems across different sectors and industries but primarily focuses on training and inference in neural networks. It is one of the most popular and in-demand frameworks and is very active in open source contribution and development.

> As of ROCm 6.1, tensorflow-rocm packages are found at https://repo.radeon.com/rocm/manylinux. Prior to ROCm 6.1, packages were found at https://pypi.org/project/tensorflow-rocm. Refer to the following version support matrix:

ROCm version	TensorFlow version
6.1.x	2.13.1, 2.14.0, 2.15.0
6.0.x	2.12, 2.13.1, 2.14.0

Pre-requisites

- Radeon software for Linux (with ROCm) must be installed.
- MIGraphX must be installed for TensorFlow to build the correct mig execution provider.

Select installation method

Option A: TensorFlow via PIP installation method

AMD recommends the PIP install method to create a TensorFlow environment when working with ROCm for machine learning development.

> **Note** The latest version of Python module numpy v2.0 is incompatible with the TensorFlow wheels for this version. Downgrade to an older version is required. Example: pip3 install numpy==1.26.4

To install TensorFlow,

Download and install the TensorFlow wheel.

```
pip3 uninstall tensorflow-rocm numpy
pip3 install https://repo.radeon.com/rocm/manylinux/rocm-rel-6.1.3/tensorflow_rocm-2.
↪15.1-cp310-cp310-manylinux_2_28_x86_64.whl numpy==1.26.4
```

Next, *verify your TensorFlow installation*.

Option B: Docker installation method

To install ROCm on bare metal, follow ROCm installation options.

Using Docker provides portability and access to a prebuilt Docker container that has been rigorously tested within AMD. This might also save compilation time and should perform as tested without facing potential installation issues.

To use Docker image with pre-installed TensorFlow

1. Enter the following command to pull the latest public TensorFlow Docker image.

   ```
   docker pull rocm/tensorflow:latest
   ```

2. Run the downloaded image.

   ```
   docker run -it --network=host --device=/dev/kfd --device=/dev/dri \
   --ipc=host --shm-size 16G --group-add video --cap-add=SYS_PTRACE \
   --security-opt seccomp=unconfined rocm/tensorflow:latest
   ```

   This will automatically download the image if it does not exist on the host. You can also pass the -v argument to mount any data directories from the host onto the container.

Next, *verify your TensorFlow installation*.

> **Note** See Tensorflow Installation for ROCm for more information.

Verify TensorFlow installation

To test the TensorFlow installation, run the container image as specified in the previous section Installing TensorFlow. Ensure you have access to the Python shell in the Docker container.

```
python3 -c 'import tensorflow' 2> /dev/null && echo 'Success' || echo 'Failure'
```

Next, *run basic TensorFlow example*.

Run basic TensorFlow example

The TensorFlow examples repository provides basic examples that exercise the framework's functionality.

The MNIST database is a collection of handwritten digits that may be used to train a Convolutional Neural Network for handwriting recognition.

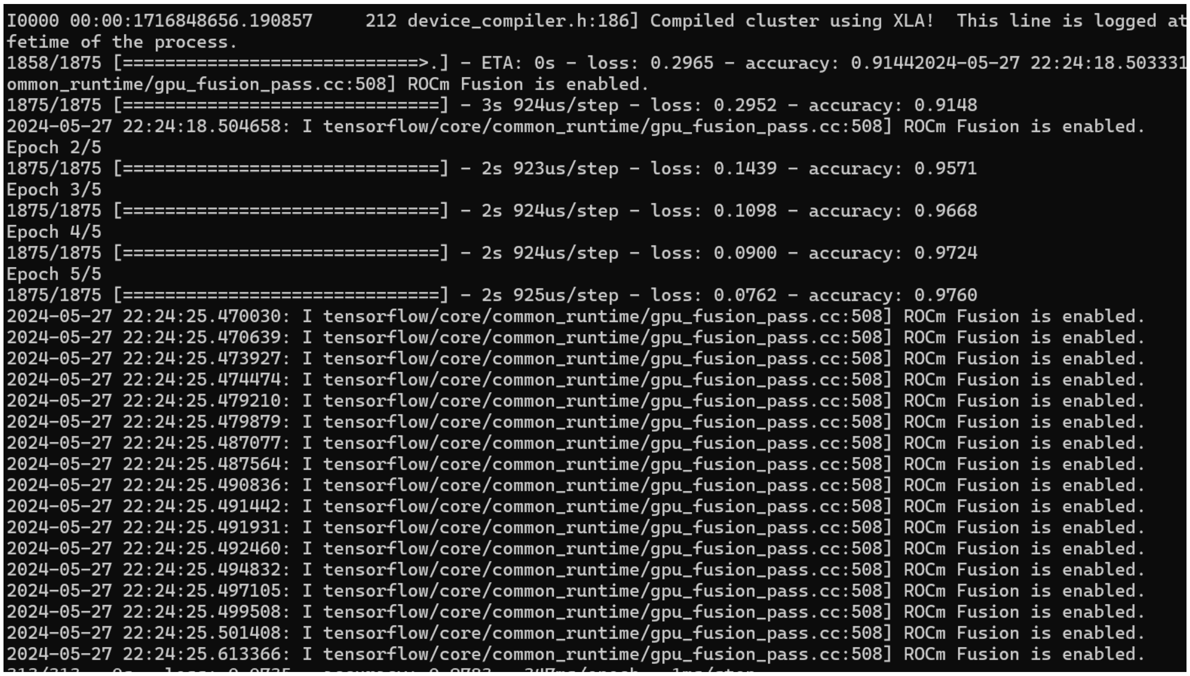
This dataset is included with your TensorFlow installation.

1. Run the following sample code to load the MNIST dataset, then train and evaluate it.

```
import tensorflow as tf
print("TensorFlow version:", tf.__version__)
mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10)
])
predictions = model(x_train[:1]).numpy()
tf.nn.softmax(predictions).numpy()
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
loss_fn(y_train[:1], predictions).numpy()
model.compile(optimizer='adam',
              loss=loss_fn,
              metrics=['accuracy'])
model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test,  y_test, verbose=2)
```

2. If successful, you should see the following output indicating the image classifier is now trained to around 98% accuracy on this dataset.



```
I0000 00:00:1716848656.190857     212 device_compiler.h:186] Compiled cluster using XLA!  This line is logged at
fetime of the process.
1858/1875 [============================>.] - ETA: 0s - loss: 0.2965 - accuracy: 0.91442024-05-27 22:24:18.503331
ommon_runtime/gpu_fusion_pass.cc:508] ROCm Fusion is enabled.
1875/1875 [==============================] - 3s 924us/step - loss: 0.2952 - accuracy: 0.9148
2024-05-27 22:24:18.504658: I tensorflow/core/common_runtime/gpu_fusion_pass.cc:508] ROCm Fusion is enabled.
Epoch 2/5
1875/1875 [==============================] - 2s 923us/step - loss: 0.1439 - accuracy: 0.9571
Epoch 3/5
1875/1875 [==============================] - 2s 924us/step - loss: 0.1098 - accuracy: 0.9668
Epoch 4/5
1875/1875 [==============================] - 2s 924us/step - loss: 0.0900 - accuracy: 0.9724
Epoch 5/5
1875/1875 [==============================] - 2s 925us/step - loss: 0.0762 - accuracy: 0.9760
2024-05-27 22:24:25.470030: I tensorflow/core/common_runtime/gpu_fusion_pass.cc:508] ROCm Fusion is enabled.
2024-05-27 22:24:25.470639: I tensorflow/core/common_runtime/gpu_fusion_pass.cc:508] ROCm Fusion is enabled.
2024-05-27 22:24:25.473927: I tensorflow/core/common_runtime/gpu_fusion_pass.cc:508] ROCm Fusion is enabled.
2024-05-27 22:24:25.474474: I tensorflow/core/common_runtime/gpu_fusion_pass.cc:508] ROCm Fusion is enabled.
2024-05-27 22:24:25.479210: I tensorflow/core/common_runtime/gpu_fusion_pass.cc:508] ROCm Fusion is enabled.
2024-05-27 22:24:25.479879: I tensorflow/core/common_runtime/gpu_fusion_pass.cc:508] ROCm Fusion is enabled.
2024-05-27 22:24:25.487077: I tensorflow/core/common_runtime/gpu_fusion_pass.cc:508] ROCm Fusion is enabled.
2024-05-27 22:24:25.487564: I tensorflow/core/common_runtime/gpu_fusion_pass.cc:508] ROCm Fusion is enabled.
2024-05-27 22:24:25.490836: I tensorflow/core/common_runtime/gpu_fusion_pass.cc:508] ROCm Fusion is enabled.
2024-05-27 22:24:25.491442: I tensorflow/core/common_runtime/gpu_fusion_pass.cc:508] ROCm Fusion is enabled.
2024-05-27 22:24:25.491931: I tensorflow/core/common_runtime/gpu_fusion_pass.cc:508] ROCm Fusion is enabled.
2024-05-27 22:24:25.492460: I tensorflow/core/common_runtime/gpu_fusion_pass.cc:508] ROCm Fusion is enabled.
2024-05-27 22:24:25.494832: I tensorflow/core/common_runtime/gpu_fusion_pass.cc:508] ROCm Fusion is enabled.
2024-05-27 22:24:25.497105: I tensorflow/core/common_runtime/gpu_fusion_pass.cc:508] ROCm Fusion is enabled.
2024-05-27 22:24:25.499508: I tensorflow/core/common_runtime/gpu_fusion_pass.cc:508] ROCm Fusion is enabled.
2024-05-27 22:24:25.501408: I tensorflow/core/common_runtime/gpu_fusion_pass.cc:508] ROCm Fusion is enabled.
2024-05-27 22:24:25.613366: I tensorflow/core/common_runtime/gpu_fusion_pass.cc:508] ROCm Fusion is enabled.
```

Environment set-up is complete, and the system is ready for use with TensorFlow to work with machine learning models, and algorithms.

2.1.5 Install MIGraphX for Radeon GPUs

MIGraphX is AMD's graph inference engine that accelerates machine learning model inference, and can be used to accelerate workloads within the Torch MIGraphX and ONNX Runtime backend frameworks.

- Torch-MIGraphX, which integrates MIGraphX with PyTorch
- MIGraphX for ONNX Runtime backend, which integrates MIGraphX with ONNX

  ONNX Runtime can be driven by either the ROCm™ Execution Provider (EP) or MIGraphX EP

Introduction to MIGraphX

MIGraphX is a graph optimizer that accelerates the inference for deep learning models. It provides C++ and Python APIs that are integrated within frameworks like Torch MIGraphX, ONNX Runtime, and other user solutions. The following process summarizes the procedures that occur under-the-hood during the optimization and real-time compilation process.

MIGraphX accelerates the Machine Learning models by leveraging several graph-level transformations and optimizations. These optimizations include:

- Operator fusion
- Arithmetic simplifications
- Dead-code elimination
- Common subexpression elimination (CSE)
- Constant propagation

When the aforementioned optimizations are applied, MIGraphX emits code for the AMD GPU by calling to MIOpen, rocBLAS, or creating HIP kernels for a particular operator. MIGraphX can also target CPUs using DNNL or ZenDNN libraries.

For more information on how to install MIGraphX, refer to AMD MIGraphX Github.

Prerequisites

- Radeon™ Software for Linux (with ROCm) is installed

Install MIGraphX

Install MIGraphX on your computer. Once the install is completed and verified, proceed to *install Torch-MIGraphX* or *MIGraphX for ONNX Runtime*.

Run the following command to install MIGraphX:

```
$ sudo apt install migraphx
```

Next, *verify MIGraphX installation*.

Verify MIGraphX installation

Perform a test to verify the MIGraphX installation.

```
migraphx-driver perf --test
```

Expected result:

```
will complete with no error
```

Next, proceed to install *Torch-MIGraphX* or *MIGraphX for ONNX Runtime* as applicable.

Install Torch-MIGraphX

Install Torch-MIGraphX using the Docker installation method, or build from source.

Option A: Docker installation

Using Docker provides portability, and access to a prebuilt Docker container that has been rigorously tested within AMD. Docker also cuts down compilation time, and should perform as expected without installation issues.

1. Clone the `torch_migraphx` repository.

   ```
   git clone https://github.com/ROCmSoftwarePlatform/torch_migraphx.git
   ```

2. Change directory to torch-migraphx.

   ```
   cd torch-migraphx/
   ```

3. Build image using the provided script.

   ```
   sudo ./build_image.sh
   ```

4. Run container.

```
sudo docker run -it --network=host --device=/dev/kfd --device=/dev/dri --group-
↪add=video --ipc=host --cap-add=SYS_PTRACE --security-opt seccomp=unconfined␣
↪torch_migraphx
```

Next, *verify the Torch-MIGraphX installation*.

Option B: Build from source

To build from source in a custom environment, refer to the torch_migraphx repository for build steps.

Next, *verify the Torch-MIGraphX installation*.

Verify Torch-MIGraphX installation

Verify if the Torch-MIGraphX installation is successful.

1. Verify if torch_migraphx can be imported as a Python module.

```
python3 -c 'import torch_migraphx' 2> /dev/null && echo 'Success' || echo 'Failure
↪'
```

2. Run unit tests.

```
pytest ./torch_migraphx/tests
```

Installation is complete and the system is able to run PyTorch through the python interface library, and scripts that invoke PyTorch inference sessions.

Install MIGraphX for ONNX Runtime

Install `onnxruntime-rocm` for ROCm by following these steps.

Prerequisites

- *MIGraphX* is installed
- Latest PyTorch ROCm release is installed

  Check Install PyTorch for ROCm for latest PIP install instructions and availability.

  Important! These instructions are validated for an Ubuntu 22.04 environment. Refer to the *OS support matrix* for more information.

To install MIGraphX for ONNX Runtime,

1. Verify that the install works correctly by performing a simple inference with MIGraphX.

```
migraphx-driver perf --test
```

2. Ensure that the half package is installed.

```
$ sudo apt install half
```

3. Install `onnxruntime-rocm` via Python PIP.

```
$ pip3 install onnxruntime-rocm -f https://repo.radeon.com/rocm/manylinux/rocm-
↪rel-6.0/
```

Next, *verify the MIGraphX for ONNX Runtime installation*.

Verify MIGraphX installation for ONNX Runtime

Verify that the install works correctly by performing a simple inference with MIGraphX.

```
$ python3
import onnxruntime as ort
ort.get_available_providers()
```

Expected result:

```
>>> import onnxruntime as ort
>>> ort.get_available_providers()
['MIGraphXExecutionProvider', 'ROCMExecutionProvider', 'CPUExecutionProvider']
```

This indicates that the `MIGraphXExecutionProvider` and `ROCMExecutionProvider` are now running on the system, and the proper ONNX Runtime package has been installed.

Installation is complete and ONNX Runtime is available through the Python interface library, as well as scripts that invoke ONNX Runtime inference sessions.

> For more information on the ONNX Runtime Python library, refer to Get started with ONNX Runtime in Python.

2.1.6 mGPU setup and configuration

Hardware and software considerations

Refer to the following hardware and software considerations to ensure optimal performance.

Hardware considerations

- **PCIe® slots** AMD recommends a system with multiple x16 (Gen 4) slots, with optimal performance achieved by provision of a 1:1 ratio between the number of x16 slots and the number of GPUs used.

  > **Note** Functionality is maintained in the instance where only one x16 slot is available, at the cost of some performance.

- **mGPU power setup** MultiGPU configurations require adequate amounts of power for all the components required. Consult AMD Radeon™ RX or AMD Radeon™ PRO for GPU specifications and graphics card power requirements.

Software considerations

There are no differences in software requirements between single-GPU and multi-GPU usage.

mGPU configuration by framework

> **Note** PyTorch, ONNX, and Tensorflow may have additional guidelines regarding mGPU configuration. Refer to official mGPU support documentation of the applicable framework for more information.

mGPU known issues and limitations

AMD has identified common errors when running ROCm™ on Radeon™ multi-GPU configuration at this time, along with the applicable recommendations.

IOMMU limitations and guidance

> For IOMMU limitations and guidance, see Issue #5: Application hangs on Multi-GPU systems.

Windows Subsystem for Linux (WSL) support

Microsoft does not currently support mGPU setup in WSL.

Simultaneous parallel compute workloads

Radeon GPUs does not support large amounts of simultaneous, parallel workloads. It is not recommended to exceed 2 simultaneous compute workloads, with the assumption that workloads are running alongside a graphics environment (eg: Linux desktop).

Recommended multi-GPU system configuration

PCIe slots connected to the GPU must have identical PCIe lane width or bifurcation settings, and support PCIe 3.0 Atomics.

Refer to How ROCm uses PCIe Atomics for more information.

> **Example**:
>
> ✓ - GPU0 PCIe x16 connection + GPU1 PCIe x16 connection
>
> ✓ - GPU0 PCIe x8 connection + GPU1 PCIe x8 connection
>
> **X** - GPU0 PCIe x16 connection + GPU1 PCIe x8 connection
>
> **Important!**
>
> - Only use PCIe slots connected by the CPU and to avoid PCIe slots connected via chipset. Refer to product-specific motherboard documentation for PCIe electrical configuration.
> - Ensure the system Power Supply Unit (PSU) has sufficient wattage to support multiple GPUs.

GPU isolation techniques

For more information, see GPU isolation techniques.

PCIe atomic operations

Some consumer grade motherboards may only support the first PCIe slot. For unexpected issues, see How ROCm uses PCIe atomics.

Errors due to GPU and PCIe configuration

When using two AMD Radeon 7900XTX GPUs, the following HIP error is observed when running PyTorch micro-benchmarking if any one of the two GPUs are connected to a non-CPU PCIe slot (PCIe on chipset):

```
RuntimeError: HIP error: the operation cannot be performed in the present state
HIP kernel errors might be asynchronously reported at some other API call, so the stacktrace below might be incorrect.
For debugging consider passing HIP_LAUNCH_BLOCKING=1.
Compile with `TORCH_USE_HIP_DSA` to enable device-side assertions.
```

Potential GPU reset with some mixed graphics and compute workloads

Working with certain mixed graphics and compute workloads may result in a GPU reset on Radeon GPUs.

Currently identified scenarios include:

- Running multiple ML workloads simultaneously while using the desktop
- Running ML workloads while simultaneously using Blender/HIP

2.2 WSL How to guide - Use ROCm on Radeon GPUs

This guide walks you through the various installation processes required to pair ROCm™ with the latest high-end AMD Radeon™ 7000 series desktop GPUs, and get started on a fully-functional environment for AI and ML development.

2.2.1 Install Radeon software for WSL with ROCm

The ROCm™ Software Stack and other Radeon™ software for Windows Subsystem for Linux (WSL) components are installed using the `amdgpu-install` script to assist you in the installation of a coherent set of stack components.

- Simplifies the installation of the AMDGPU stack by encapsulating the distribution specific package installation logic and by using command line options that allows you to specify the:
  - Usecase of the AMDGPU stack to be installed (WSL)
  - Combination of components (Pro stack, or user selection)
- Performs post-install checks to verify whether the installation was performed successfully.
- Installs the uninstallation script to allow you to remove the whole AMDGPU stack from the system by using a single command.

The script is provided by the installer package. See *Compatibility matrices* for support information.

Prerequisites

- **WSL is installed** Ensure that WSL is installed before proceeding with ROCm installation.

  To install WSL, refer to Windows Subsystem for Linux Documentation.

- **Compatible Radeon™ Software for Windows driver is installed** WSL requires installation of the following Windows driver.

  To install the compatible driver, refer to AMD Software: Adrenalin Edition™ 24.6.1 for WSL 2.

Install AMD unified driver package repositories and installer script

Download and install the `amdgpu-install` script on the system.

Enter the following commands to install the installer script for the latest compatible Ubuntu® version:

```
sudo apt update
wget https://repo.radeon.com/amdgpu-install/6.1.3/ubuntu/jammy/amdgpu-install_6.1.60103-1_all.deb
sudo apt install ./amdgpu-install_6.1.60103-1_all.deb
```

Install AMD unified kernel-mode GPU driver, ROCm, and graphics

After the Unified Driver Deb Package repositories are installed, run the installer script with appropriate `--usecase` parameters to install the driver components.

AMD recommends installing the WSL usecase by default.

Enter the following command to display a list of available usecases:

```
sudo amdgpu-install --list-usecase
```

WSL usecase

AMD recommends installing the WSL usecase by default.

Run the following command to install open source graphics and ROCm.

```
amdgpu-install -y --usecase=wsl,rocm --no-dkms
```

> Watch for output warning or errors indicating an unsuccessful driver installation.
>
> **Note:** The `-y` option installs non-interactively. This step may take several minutes, depending on internet connection and system speed.

Next, *run a post-install verification check*.

Post-install verification check

Run a post-installation check to verify that the installation is complete:

Check if the GPU is listed as an agent.

```
rocminfo
```

Expected result:

```
[...]
*******
Agent 2
*******
  Name:                    gfx1100
  Marketing Name:          Radeon RX 7900 XTX
  Vendor Name:             AMD
  [...]
[...]
```

Uninstall ROCm

Run the following command to uninstall the ROCm software stack and other Radeon software for Linux components:

```
sudo amdgpu-uninstall
```

Upgrade to newer versions of Radeon software for Linux

The recommended method to upgrade is to uninstall, followed by an install.

> Radeon Software for Linux does not support in-place upgrades.

2.2.2 Install PyTorch for ROCm

Refer to this section for the recommended PyTorch via PIP installation method, as well as Docker-based installation.

Install methods

AMD recommends the PIP install method to create a PyTorch environment when working with ROCm™ for machine learning development.

Using Docker provides portability, and access to a prebuilt Docker container that has been rigorously tested within AMD. Docker also cuts down compilation time, and should perform as expected without installation issues.

Option A: PyTorch via PIP installation

AMD recommends the PIP install method to create a PyTorch environment when working with ROCm™ for machine learning development.

> Check Pytorch.org for latest PIP install instructions and availability. See *Compatibility matrices* for support information.
>
> **Note** The latest version of Python module numpy v2.0 is incompatible with the torch wheels for this version. Downgrade to an older version is required. Example: pip3 install numpy==1.26.4

Install PyTorch via PIP

1. Enter the following command to unpack and begin set up.

```
sudo apt install python3-pip -y
```

2. Enter this command to update the pip wheel.

```
pip3 install --upgrade pip wheel
```

3. Enter this command to install Torch and Torchvision for ROCm AMD GPU support.

```
wget https://repo.radeon.com/rocm/manylinux/rocm-rel-6.1.3/torch-2.1.2%2Brocm6.1.
↪3-cp310-cp310-linux_x86_64.whl
wget https://repo.radeon.com/rocm/manylinux/rocm-rel-6.1.3/torchvision-0.16.1
↪%2Brocm6.1.3-cp310-cp310-linux_x86_64.whl
wget https://repo.radeon.com/rocm/manylinux/rocm-rel-6.1.3/pytorch_triton_rocm-2.
↪1.0%2Brocm6.1.3.4d510c3a44-cp310-cp310-linux_x86_64.whl
pip3 uninstall torch torchvision pytorch-triton-rocm numpy
pip3 install torch-2.1.2+rocm6.1.3-cp310-cp310-linux_x86_64.whl torchvision-0.16.
↪1+rocm6.1.3-cp310-cp310-linux_x86_64.whl pytorch_triton_rocm-2.1.0+rocm6.1.3.
↪4d510c3a44-cp310-cp310-linux_x86_64.whl numpy==1.26.4
```

   This may take several minutes.

   Important! AMD recommends proceeding with ROCm WHLs available at repo.radeon.com. The ROCm WHLs available at PyTorch.org are not tested extensively by AMD as the WHLs change regularly when the nightly builds are updated.

4. Update to WSL compatible runtime lib.

```
location=`pip show torch | grep Location | awk -F ": " '{print $2}'`
cd ${location}/torch/lib/
rm libhsa-runtime64.so*
cp /opt/rocm/lib/libhsa-runtime64.so.1.2 libhsa-runtime64.so
```

Next, *verify your PyTorch installation*.

Option B: Docker installation

Using Docker provides portability, and access to a prebuilt Docker container that has been rigorously tested within AMD. Docker also cuts down compilation time, and should perform as expected without installation issues.

Prerequisites to install PyTorch using Docker

Docker for Ubuntu® must be installed.

To install Docker for Ubuntu, enter the following command:

```
sudo apt install docker.io
```

Use Docker image with pre-installed PyTorch

Follow these steps for installing using a Docker image.

1. Enter the following command to pull the public PyTorch Docker image.

   ```
   sudo docker pull rocm/pytorch:rocm6.1.3_ubuntu22.04_py3.10_pytorch_release-2.1.2
   ```

   Optional: You can also download a specific and supported configuration with different user-space ROCm versions, PyTorch versions, and supported operating systems.

   Refer to hub.docker.com/r/rocm/pytorch to download the PyTorch Docker image.

2. Start a Docker container using the downloaded image.

   ```
   sudo docker run -it \
   --cap-add=SYS_PTRACE  \
   --security-opt seccomp=unconfined \
   --ipc=host \
   --shm-size 8G \
   --device=/dev/dxg -v /usr/lib/wsl/lib/libdxcore.so:/usr/lib/libdxcore.so -v /opt/
   ↪rocm/lib/libhsa-runtime64.so.1:/opt/rocm/lib/libhsa-runtime64.so.1  \
   rocm/pytorch:rocm6.1.3_ubuntu22.04_py3.10_pytorch_release-2.1.2
   ```

   This will automatically download the image if it does not exist on the host. You can also pass the -v argument to mount any data directories from the host onto the container.

Next, *verify the PyTorch installation*.

See PyTorch Installation for ROCm for more information.

Verify PyTorch installation

Confirm if PyTorch is correctly installed.

1. Verify if Pytorch is installed and detecting the GPU compute device.

   ```
   python3 -c 'import torch' 2> /dev/null && echo 'Success' || echo 'Failure'
   ```

   Expected result:

   ```
   Success
   ```

2. Enter command to test if the GPU is available.

```
python3 -c 'import torch; print(torch.cuda.is_available())'
```

Expected result:

```
True
```

3. Enter command to display installed GPU device name.

```
python3 -c "import torch; print(f'device name [0]:', torch.cuda.get_device_
↪name(0))"
```

Expected result: Example: *device name [0]: Radeon RX 7900 XTX*

```
device name [0]: <Supported AMD GPU>
```

4. Enter command to display component information within the current PyTorch environment.

```
python3 -m torch.utils.collect_env
```

Expected result:

```
PyTorch version

ROCM used to build PyTorch

OS

Is CUDA available

GPU model and configuration

HIP runtime version

MIOpen runtime version
```

Environment set-up is complete, and the system is ready for use with PyTorch to work with machine learning models, and algorithms.

CHAPTER

THREE

USECASES

Refer to the applicable guides to optimize specific usecase performance.

3.1 vLLM

Refer to the applicable guides to optimize vLLM usecase performance.

3.1.1 vLLM Docker image for Llama2 and Llama3

Virtual Large Language Model (vLLM) is a fast and easy-to-use library for LLM inference and serving.

Llama2 and Llama3 support is enabled via a vLLM Docker image that must be built separately (in addition to ROCm) for the current release.

For additional information, visit the AMD vLLM GitHub page.

> Note that this is a benchmarking demo/example. Installation for other vLLM models/configurations may differ.

Prerequisites

- GitHub is authenticated.

Additional information

- AMD recommends 40GB GPU for 70B usecases.Ensure that your GPU has enough VRAM for the chosen model.
- This example highlights use of the AMD vLLM Docker using Llama-3 70B with GPTQ quantization (as shown at Computex).However, performance is not limited to this specific Hugging Face model, and other vLLM supported models can also be used.

Installation steps

Follow these steps to build a vLLM Docker image and start using Llama2 and Llama3.

1. Clone the ROCm/vllm repository.

```
git clone -b vllm0.4.1_llama_70b_gptq https://github.com/ROCm/vllm.git
```

2. Change directory to vLLM, and build Docker image.

```
sudo docker build -f Dockerfile.rocm -t <image_name> .
```

   Note

   - The Docker `image_name` is user defined. Ensure to name your Docker using this value. **Example**: vllm0.4.1_rocm6.1.1_ubuntu20.04_py3.9_image
   - **Optional**: Map the vllm directory from the host to the Docker container.

3. Start the Docker container.

```
sudo docker run -it --privileged --device=/dev/kfd --device=/dev/dri --
↪network=host --group-add sudo -w /root/workspace --name <container_name> <image_
↪name> /bin/bash
```

   Note The `container_name` is user defined. Ensure to name your Docker using this value.

   Example: vllm0.4.1_rocm6.1.1_ubuntu20.04_py3.9_container

4. Clone the vLLM GitHub repository within the Docker container.

   This step is not necessary if mapped in Step 2.

```
git clone -b vllm0.4.1_llama_70b_gptq https://github.com/ROCm/vllm.git
```

5. Clone the Hugging Face GitHub repository within the Docker container.

```
git lfs clone https://huggingface.co/TechxGenus/Meta-Llama-3-70B-Instruct-GPTQ
```

6. Run benchmarks within the Docker container.

```
python3 vllm/benchmarks/benchmark_latency.py --model /root/workspace/Meta-Llama-3-
↪70B-Instruct-GPTQ --batch-size 1 --input-len 1024 --output-len 1024
```

   Note Ensure that the model is downloaded and vLLM checkout is set to your current directory within the container described in Step 3.

Your environment is set up to use Llama2 and Llama3.

CHAPTER

FOUR

COMPATIBILITY MATRICES

This section provides information on the compatibility of ROCm™ components, Radeon™ GPUs, and the Radeon Software for Linux® version (Kernel Fusion Driver) and Windows Subsystem for Linux (WSL).

Note: To rollback support matrices and install instructions for previous versions, click **Version List** located at the top-right corner of the screen, or select the version (v:) menu on the bottom-left.

Linux

WSL

Linux Compatibility

WSL Compatibility

4.1 Linux support matrices by ROCm version

4.1.1 ROCm 6.1.3

Compatible OS, GPU, and framework support matrices for the latest ROCm release.

> To rollback support matrices and install instructions for previous versions, click **Version List** located at the top-right corner of the screen, or select the version (v:) menu on the bottom-left.

OS support matrix

OS	Kernel	Supported
Ubuntu® 22.04.4 Desktop Version with HWE	Ubuntu kernel 6.5	Yes

GPU support matrix

ROCm Version	Radeon™ Software for Linux® Version	Supported AMD Radeon™ Hardware
6.1.3	24.10.3	AMD Radeon RX 7900 XTXAMD Radeon RX 7900 XTAMD Radeon RX 7900 GREAMD Radeon PRO W7900AMD Radeon PRO W7900DSAMD Radeon PRO W7800

Framework + ROCm support matrices

View the ROCm support matrices for PyTorch, ONNX, and TensorFlow frameworks.

PyTorch + ROCm support matrix

PyTorch Version	ROCm Version	Comments
2.1.2	6.1.3	Official production support. Available from AMD.com.
2.5+/Nightly	6.1	Available from PyTorch.org nightly builds, not tested extensively by AMD.
2.3/Stable	6.0	Not supported for Radeon 7000 series.

AI Data Operations

- FP32
- FP16
- Mixed precision (FP32/FP16)
- INT8

ONNX + ROCm support matrix

ONNX Version	ROCm Version	Comments
1.17	6.1.3	Official production support. Available from AMD.com.

AI Data Operations

- FP32
- FP16
- INT8 (MIGraphX)
- Mixed precision (FP32/FP16)

Note Refer to Installation Instructions to Get Started with ONNX Runtime for more information.

TensorFlow + ROCm support matrix

TensorFlow Version	ROCm Version	Comments
2.15	6.1.3	Official production support. Available from AMD.com.

AI Data Operations

- FP32
- FP16

4.1.2 Docker support matrix

See Docker Image Support Matrix for the latest version of the software support matrices for ROCm container releases.

4.2 Compatibility matrices (WSL)

4.2.1 ROCm 6.1.3

This section provides information on the compatibility of ROCm™ components, Radeon™ GPUs, and the Radeon Software for Windows Subsystem for Linux® (WSL).

> To rollback support matrices and install instructions for previous versions, click **Version List** located at the top-right corner of the screen, or select the version (v:) menu on the bottom-left.

OS support matrix

OS	Kernel	Supported
Ubuntu 22.04 or Ubuntu 22.04 LTS	WSL2-Linux-Kernel 5.15	Yes

> **Note** AMD recommends using Ubuntu 22.04. Refer to How to install Linux on Windows with WSL for up-to-date OS compatibility information.

GPU support matrix

ROCm Version	Radeon™ Software for Linux® Version	Radeon™ Software for Windows Version	Supported AMD Radeon™ Hardware
6.1.3	24.10.3	AMD Software: Adrenalin Edition™ 24.6.1 for WSL 2	AMD Radeon RX 7900 XTXAMD Radeon RX 7900 XTAMD Radeon RX 7900 GREAMD Radeon PRO W7900AMD Radeon PRO W7900DSAMD Radeon PRO W7800

Framework + ROCm support matrices

View the ROCm support matrices for PyTorch.

PyTorch + ROCm support matrix

PyTorch Version	ROCm Version	Comments
2.1.2	6.1.3	Official production support. Available from AMD.com.
2.5+/Nightly	6.1	Available from PyTorch.org nightly builds, not tested extensively by AMD.
2.3/Stable	6.0	Not supported for Radeon 7000 series.

AI Data Operations

- FP32
- FP16
- Mixed precision (FP32/FP16)
- INT8

4.2.2 Docker support matrix

See Docker Image Support Matrix for the latest version of the software support matrices for ROCm container releases.

CHAPTER

FIVE

LIMITATIONS

This section provides information on software and configuration limitations.

> **Important!** Radeon™ PRO Series graphics cards are not designed nor recommended for datacenter usage. Use in a datacenter setting may adversely affect manageability, efficiency, reliability, and/or performance. GD-239.
>
> **Important!** ROCm is not officially supported on any mobile SKUs.

5.1 Multi-GPU configuration

See *mGPU known issues and limitations*.

5.2 Windows Subsystem for Linux (WSL)

Due to limited validation of ROCm™ on Radeon™ WSL configuration at this time, common errors and applicable recommendations are identified.

> **Important!** ROCm 6.1.3 release is limited to preview support for WSL configuration.
>
> At this time, only a limited amount of validation has been performed. AMD recommends only proceeding with advanced know-how and at user discretion.
>
> Visit the AI community to share feedback, and Report a bug if you find any issues.
>
> **Note** Refer to Microsoft WSL Documentation for latest information on WSL support for mGPU configurations.

5.2.1 ROCm support in WSL environments

ROCm-smi support

Due to WSL architectural limitations for native Linux User Kernel Interface (UKI), rocm-smi is not supported.

Issue	Limitations
UKI does not currently support rocm-smi	**No current support for:**Active compute processesGPU utilizationModifiable state features

ROCm-profiler support

Not currently supported.

Debugger

Not currently supported.

5.2.2 Running PyTorch in virtual environments

Running PyTorch in virtual environments requires a manual libhsa-runtime64.so update.

When using the WSL usecase and hsa-runtime-rocr4wsl-amdgpu package (installed with PyTorch wheels), users are required to update to a WSL compatible runtime lib.

Solution:

Enter the following commands:

```
location=`pip show torch | grep Location | awk -F ": " '{print $2}'`
cd ${location}/torch/lib/
rm libhsa-runtime64.so*
cp /opt/rocm/lib/libhsa-runtime64.so.1.2 libhsa-runtime64.so
```

5.3 6.1.3 release known issues

- Radeon GPUs do not support large amounts of simultaneous parallel workloads. We do not recommend exceeding 2 simultaneous compute workloads. This assumes workloads are running alongside a graphics environment (eg: Linux desktop).
- Intermittent gpureset errors may be seen with Automatic 1111 webUI with IOMMU enabled. Refer to the AMD community knowledge base for suggested resolutions.
- ROCm debugger is unstable and not fully supported in this release.
- "Automatic suspend state when idle" is not recommended when running AI workloads.
- Connecting monitors to multiple GPUs on an Intel Sapphire Rapids powered system may cause displays to not display correctly. We recommend connecting all the monitors into one GPU. For configurations where monitors must be plugged into multiple GPUs, please add the following to your grub file: "intel_iommu=on iommu=on"

5.3.1 WSL specific issues

- Some long running rocsparse kernels may trigger a TDR.

CHAPTER

SIX

AI COMMUNITY

Want to share your experiences, find answers, or contribute to resolving issues?

Explore the AMD AI Community Forum, where you will find a like-minded community, passionate about all things AI!

CHAPTER

SEVEN

REPORT A BUG

Found a defect? Report issues through ROCm GitLab, and contribute to improving our user experience.