

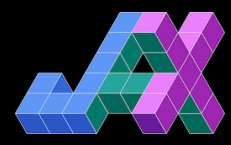


ML and AI on AMD GPUs

Presenter: Sidafa Conde

ML ECOSYSTEM FOR AMD GPUS

SOURCE & BINARY implementations available upstream!



ONNX
RUNTIME

MIOPEN | RCCL | MIVisionX | MIGraphX

DEEPSPEED | AITemplate | openAI Triton | CuPy | XGBoost | hip-Python

flash-attention | vLLM | bitsandbytes | MPI4Py



AMD
ROCm

AMD
INSTINCT

What is PyTorch

- PyTorch is a Python™ package that provides two high-level features:
 - Tensor computation (like NumPy) with strong GPU acceleration
 - Deep neural networks built on a tape-based autograd system
- You can reuse your favorite Python™ packages such as NumPy, SciPy, and Cython to extend PyTorch when needed

source: [pytorch github repo](#)

PyTorch

At a granular level, PyTorch is a library that consists of the following components:

Component	Description
torch	A Tensor library like NumPy, with strong GPU support
torch.autograd	A tape-based automatic differentiation library that supports all differentiable Tensor operations in torch
torch.jit	A compilation stack (TorchScript) to create serializable and optimizable models from PyTorch code
torch.nn	A neural networks library deeply integrated with autograd designed for maximum flexibility
torch.multiprocessing	Python multiprocessing, but with magical memory sharing of torch Tensors across processes. Useful for data loading and Hogwild training
torch.utils	DataLoader and other utility functions for convenience

Usually, PyTorch is used either as:

- A replacement for NumPy to use the power of GPUs.
- A deep learning research platform that provides maximum flexibility and speed.

What is JAX



- JAX is a Python™ library for accelerator-oriented array computation and program transformation, designed for HPC and large-scale ML
- It can automatically **differentiate** native Python and NumPy function
- It uses the Accelerated Linear Algebra ([XLA](#)) compiler to compile and scale your NumPy programs on TPUs, GPUs, and other hardware accelerators
- It is a research project, not an official Google product. Expect sharp edges

source: [jax github repo](#)

JAX vs. NumPy

Key concepts:

- JAX provides a NumPy-inspired interface for convenience.
- Through duck-typing, JAX arrays can often be used as drop-in replacements of NumPy arrays.
- Unlike NumPy arrays, JAX arrays are always immutable.

NumPy, lax & XLA: JAX API layering

Key concepts:

- `jax.numpy` is a high-level wrapper that provides a familiar interface.
- `jax.lax` is a lower-level API that is stricter and often more powerful.
- All JAX operations are implemented in terms of operations in [XLA](#) – the Accelerated Linear Algebra compiler.

To JIT or not to JIT

Key concepts:

- By default JAX executes operations one at a time, in sequence.
- Using a just-in-time (JIT) compilation decorator, sequences of operations can be optimized together and run at once.
- Not all JAX code can be JIT compiled, as it requires array shapes to be static & known at compile time.

source: [thinking_in_jax](#)

What is TensorFlow

- TensorFlow is an end-to-end open-source platform for machine learning
- It has a comprehensive, flexible ecosystem of **tools, libraries, and community resources** to easily build and deploy ML-powered applications
- TensorFlow was originally developed by researchers and engineers working within the Machine Intelligence team at Google Brain
- It provides stable Python and C++ APIs, as well as a non-guaranteed backward compatible API for other languages.

source: [tensorflow github repo](#)



The TensorFlow tutorials are written as Jupyter notebooks and run directly in Google Colab—a hosted notebook environment that requires no setup. At the top of each tutorial, you'll see a **Run in Google Colab** button. Click the button to open the notebook and run the code yourself.

For beginners

The best place to start is with the user-friendly Keras sequential API. Build models by plugging together building blocks. After these tutorials, read the [Keras guide](#).

Beginner quickstart

This "Hello, World!" notebook shows the Keras Sequential API and `model.fit`.

Keras basics

This notebook collection demonstrates basic machine learning tasks using Keras.

Load data

These tutorials use `tf.data` to load various data formats and build input pipelines.

For experts

The Keras functional and subclassing APIs provide a define-by-run interface for customization and advanced research. Build your model, then write the forward and backward pass. Create custom layers, activations, and training loops.

Advanced quickstart

This "Hello, World!" notebook uses the Keras subclassing API and a custom training loop.

Customization

This notebook collection shows how to build custom layers and training loops in TensorFlow.

Distributed training

Distribute your model training across multiple GPUs, multiple machines or TPUs.

source: [tensorflow tutorials](#)

What is ONNX

- Open Neural Network Exchange (**ONNX**) provides an open source format for AI models, both deep learning and traditional ML.
- It defines an extensible computation graph model, as well as definitions of built-in operators and standard data types.

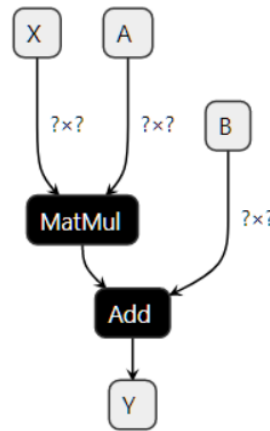


ONNX Concepts

ONNX can be compared to a programming language specialized in mathematical functions. It defines all the necessary operations a machine learning model needs to implement its inference function with this language. A linear regression could be represented in the following way:

```
def onnx_linear_regressor(X):  
    "ONNX code for a linear regression"  
    return onnx.Add(onnx.MatMul(X, coefficients), bias)
```

This example is very similar to an expression a developer could write in Python. It can be also represented as a graph that shows step-by-step how to transform the features to get a prediction. That's why a machine-learning model implemented with ONNX is often referenced as an **ONNX graph**.



Use ONNX

- [Documentation of ONNX Python Package](#)
- [Tutorials for creating ONNX models](#)
- [Pre-trained ONNX models](#)

source: [onnx github repo](#)

source: [onnx documentation](#)

What is CuPy



- NumPy is a python interface to optimized routines written in C that provide arrays, multi-dimensional arrays and common numerical operations on them. These are much faster than operating on Python lists
- SciPy provides fundamental algorithms common in scientific and numerical computing. The underlying code is a mixture of Fortran, C and C++
- CuPy is a NumPy/SciPy-compatible array library for GPU-accelerated computing with Python
- CuPy acts as a drop-in replacement to run existing NumPy/SciPy code on NVIDIA CUDA or AMD ROCm™ platforms
- CuPy provides the ndarray, sparse matrices, and the associated routines for GPU devices, most having the same API as NumPy and SciPy.
- CuPy provides interfaces to GPU optimized libraries such as rocBLAS, rocSPARSE, rocFFT, and RCCL

source: [cupy documentation](#)

click [here](#) for differences between CuPy and NumPy

CuPy functions

CuPy vs NumPy API

CuPy-specific functions

Comparison Table

Here is a list of NumPy / SciPy APIs and its corresponding CuPy implementations.

- in CuPy column denotes that CuPy implementation is not provided yet. We welcome contributions for these functions.

NumPy / CuPy APIs

Module-Level

NumPy	CuPy
<code>numpy.DataSource</code>	<code>cupy.DataSource</code> (alias of <code>numpy.DataSource</code>)
<code>numpy.ScalarType</code>	-
<code>numpy.abs</code>	<code>cupy.abs</code>
<code>numpy.absolute</code>	<code>cupy.absolute</code>
<code>numpy.add</code>	<code>cupy.add</code>
<code>numpy.all</code>	<code>cupy.all</code>
<code>numpy.allclose</code>	<code>cupy.allclose</code>

full list here: [cupy documentation](#)

CuPy-specific functions

CuPy-specific functions are placed under `cupyx` namespace.

<code>cupyx.rsqrt</code>	Returns the reciprocal square root.
<code>cupyx.scatter_add</code> (a, slices, value)	Adds given values to specified elements of an array.
<code>cupyx.scatter_max</code> (a, slices, value)	Stores a maximum value of elements specified by indices to an array.
<code>cupyx.scatter_min</code> (a, slices, value)	Stores a minimum value of elements specified by indices to an array.
<code>cupyx.empty_pinned</code> (shape[, dtype, order])	Returns a new, uninitialized NumPy array with the given shape and dtype.
<code>cupyx.empty_like_pinned</code> (a[, dtype, order, ...])	Returns a new, uninitialized NumPy array with the same shape and dtype as those of the given array.

full list here: [cupy documentation](#)

CuPy-Xarray: Xarray on GPUs

- Xarray: Python™ library to work with labelled multi-dimensional arrays
 - Popular for applications where multi-dimensional data needs to be handled (such as climate modeling)
 - Built on top of NumPy
 - Has built-in support for NetCDF
 - Can wrap custom duck array objects (i.e. NumPy-like arrays) that follow specific protocols.
- When used together, Xarray and CuPy can provide an easy way to take advantage of GPU acceleration for scientific computing tasks.
- CuPy-Xarray provides an interface for using CuPy in Xarray, providing accessors on the Xarray objects.
 - CuPy-Xarray relies on an existing CuPy installation, install CuPy first
- Cupy-Xarray github repo: <https://github.com/xarray-contrib/cupy-xarray>
 - Install with `pip install cupy-xarray --no-deps` after installing CuPy
- Issue with dask: <https://github.com/xarray-contrib/cupy-xarray/pull/62>
 - Did not make it into the latest release
 - Make sure to install dask with `pip install dask`

source: [cupy-xarray documentation](#)

What is MPI4Py

- The Message Passing Interface (MPI) is a standardized and portable message-passing system designed to function on a wide variety of parallel computers
- The MPI standard defines the syntax and semantics of library routines and allows users to write portable programs in the main scientific programming languages (Fortran, C, or C++).
- MPI for Python™ provides (MPI4Py) MPI bindings for the Python™ programming language, allowing any Python program to exploit multiple processors across multiple nodes.
- MPI4Py can send data directly from one GPU to another GPU by using GPU-aware MPI.
- MPI4Py can be configured to use any MPI implementation

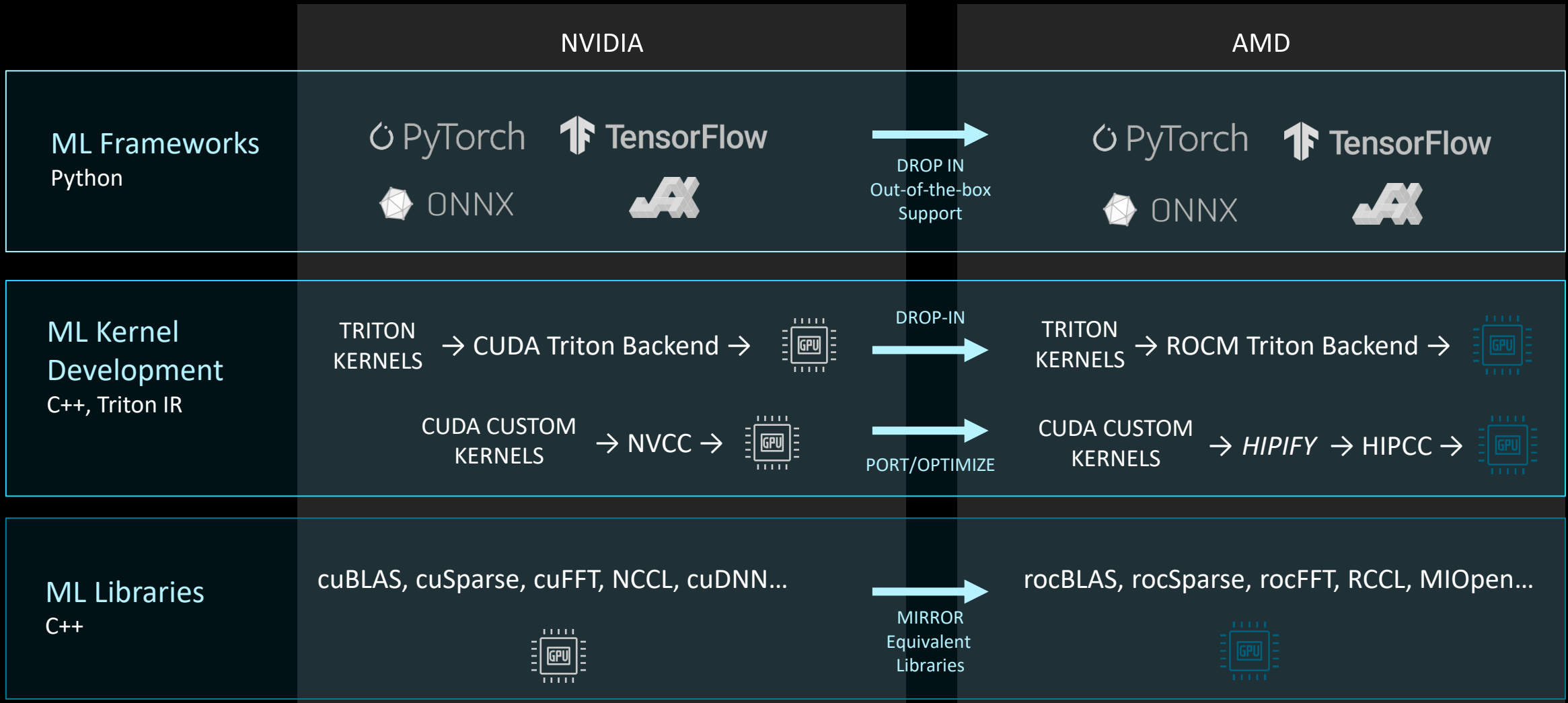
source: [mpi4py documentation](#)

ML ECOSYSTEM FOR AMD GPUS

Frameworks	Libraries & Tools	LLM Runtimes & Serving	Platform & Deployment
Tensorflow	MIOpen (primitives)	Ollama, LM Studio (local LLM hosting & UI)	ROCm – open-source driver & runtime
PyTorch	RCCL (collectives)	LangChain (orchestration)	AMD Instinct – MI200, MI250(X), MI300(X)
JAX	rocBLAS, rocFFT, rocSOLVER, rocPRIM, rocRAND	Triton Inference Server, KServe	AMD Radeon – RDNA2/RDNA3 (Navi-based) desktop & edge GPUs
ONNX Runtime	HIP & hipify (CUDA→HIP porting)		Docker/Singularity containers
MXNet	MIVisionX & MIGraphX (CV)		Slurm, MPI, Kubernetes scheduling
Horovod (distributed training)	DeepSpeed & AI Template		
Hugging Face Transformers & Diffusers	OpenAI Triton & CuPy		
	XGBoost & HIP-Python		
	FlashAttention, vLLM & BitsAndBytes		
	rocprofiler, roctracer (profiling / tracing) or rocprofiler-sdk, rocprofiler-systems, rocprofiler-compute		
	AMD uProf (system-level profiling)		

PORTING YOUR ML APPLICATION

NO CHANGES REQUIRED IN MOST CASES



PORTING YOUR ML APPLICATION

API COMPATIBLE LIBRARIES





CUDA Library	ROCm Library	Description
cuBLAS	rocBLAS	Basic Linear Algebra Subroutines
cuFFT	rocFFT	Fast Fourier Transform Library
cuSPARSE	rocSPARSE	Sparse BLAS + SPMV
cuSolver	rocSolver	LAPACK Library
AmgX	rocALUTION	Algebraic Multi-Grid Accelerated Linear Solvers
Thrust	hipThrust	C++ parallel algorithms library
CUB	rocPRIM	Low Level Optimized Parallel Primitives
cuDNN	MIOpen	Deep learning Solver Library
cuRAND	rocRAND	Random Number Generator Library
NCCL	RCCL	Communications Primitives Library based on the MPI equivalents
cuTensor	hipTensor	Library to accelerate tensor primitives

Linear Algebra Libraries supporting both AMD and NVIDIA GPUs

Eigen	C++ template library for linear algebra
MAGMA	Dense linear algebra library on GPU and Multicore Architectures
SuperLU_DIST	Direct solution of large, sparse, nonsymmetric systems of linear equation
HYPRE	Scalable Linear Solvers and Multigrid Methods
ELPA	Highly efficient and highly scalable direct eigensolvers for symmetric (Hermitian) matrices.
...	

WHERE CAN I GET AI FRAMEWORKS FROM?

BINARY AND SOURCE DISTRIBUTIONS AVAILABLE

	Source	Container	PIP wheel
 PyTorch	PyTorch GitHub	Docker Hub	pytorch.org
JAX	JAX GitHub	Docker Hub	ROCm GitHub
 TensorFlow	TensorFlow GitHub	Docker Hub	pypi.org
 ONNX RUNTIME	ONNX-RT GitHub	Docker Hub	onnxruntime.ai
DeepSpeed	DeepSpeed GitHub	Docker Hub	deepspeed.ai
 CuPy	CuPy Github	Docker Hub	cupy.dev

Not an
NVIDIA
product →

- ▲ Source builds can sometimes be.....tricky
- ▲ Leveraging containers or pre-built wheel files for Python installs is recommended if possible

WHERE CAN I GET AI FRAMEWORKS FROM?

ROCm GITHUB HOSTS MOST AMD GPU PORTS

- ▶ <https://github.com/ROCm>
- ▶ <https://github.com/ROCm/pytorch>
- ▶ <https://github.com/ROCm/jax>
- ▶ <https://github.com/ROCm/tensorflow-upstream>
- ▶ <https://github.com/ROCm/cupy>

Not all of the above branches are updated with the same frequency, please make sure you use the release of branch tag that is most appropriate to your needs and ROCm version

The screenshot shows the GitHub repository for AMD ROCm Software. The repository name is "AMD ROCm™ Software" and it has 1.2k followers. The repository is public and is part of AMD. The README file is visible, containing the following text:

AMD ROCm™ Software

AMD ROCm software is AMD's Open Source stack for GPU computation.

To learn more about ROCm, check out our [Documentation](#), [Examples](#), and [Developer Hub](#).

If you have questions or need help, reach out to us on GitHub.

Below the README, there are four popular repositories listed:

- ROCm**: AMD ROCm™ Software - GitHub Home. 5.2k stars, 427 forks.
- hip**: HIP: C++ Heterogeneous-Compute Interface for Portability. 4k stars, 550 forks.
- MIOpen**: AMD's Machine Intelligence Library. 1.1k stars, 248 forks.
- tensorflow-upstream**: Forked from tensorflow/tensorflow. TensorFlow ROCm port. 690 stars, 99 forks.

CONTAINER -ROCm ML FRAMEWORK IMAGES

ROCm DOCKERHUB PROVIDES SEVERAL IMAGES READY TO USE

- There are instructions on DockerHub on what commands to use

- Docker command:

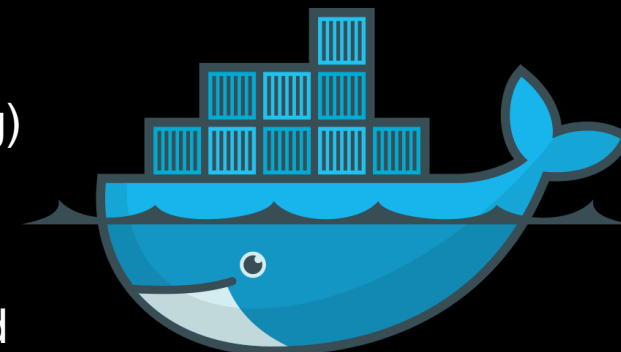
```
sudo docker run -it \
  --network=host \
  --device=/dev/kfd \
  --device=/dev/dri \
  --group-add=video \
  --ipc=host \
  --cap-add=SYS_PTRACE \
  --security-opt seccomp=unconfined \
  rocm/<FRAMEWORK>:<TAG>
```

Allow container to access network (dist. learning)

Allow access to the GPU drivers

Security options - may need to be expanded depending on the host system configuration

Replace <FRAMEWORK>:<TAG> with your choice—e.g. `rocm/pytorch:latest` or `rocm/tensorflow:5.4.0`.



- Docker images with different tags/versions are available on ROCm DockerHub

WHEEL – PRE-BUILT WITH ROCm SUPPORT

ROCm DOCKERHUB PROVIDES SEVERAL IMAGES READY TO USE

Prerequisites

- System dependencies need to be met:
 - GPU drivers
 - ROCm libraries, RCCL, and MIOpen should be installed:
 - rocm-dev
 - hiplibsdk
 - ml-sdk
- ROCm and Python versions must match the wheel file

ROCm Docs for AI

- [Use ROCm for AI](#)
- [AI Tutorials](#)
- [ROCm Libraries](#)
- [ROCm for AI Training](#)
- [ROCm for AI Inference](#)
- [ROCm for AI Inference Optimization](#)
- [Deep Learning Frameworks](#)

Example for PyTorch install with wheel

- PyTorch official docs: <https://pytorch.org/>

NOTE: Latest PyTorch requires Python 3.9 or later.

PyTorch Build	Stable (2.7.0)			Preview (Nightly)
Your OS	Linux		Mac	Windows
Package	Conda	Pip		LibTorch
Language	Python			C++ / Java
Compute Platform	CUDA 11.8	CUDA 12.6	CUDA 12.8	ROCm 6.3
	CPU			
Run this Command:	<pre>pip3 install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/rocm6.3</pre>			

Choice for a stable or a preview build

Supporting ROCm version

Suggested install command

- More combinations available! Check here <https://download.pytorch.org/whl/torch/>
 - E.g. PyTorch 2.7.0 with ROCm 6.3
 - Older versions
- Even more combinations available in <https://repo.radeon.com/rocm/manylinux>.

PyTorch wheel install – sys Python™

- Native install from PyTorch Python™ wheels

Where do we want to install things - don't use your home folder!

Package install version can mix the PyTorch version as well as the ROCm it was built against.

This is where the wheel will be pulled from

```
pip3 install -t $wd/pip-installs --pre torch==2.7.0+rocm6.3 --index-url https://download.pytorch.org/whl/
```

```
Looking in indexes: https://download.pytorch.org/whl/
```

```
Collecting torch==2.7.0+rocm6.3
```

```
Downloading https://download.pytorch.org/whl/rocm6.3/torch-2.7.0%2Brocm6.3-cp310-cp310-manylinux_2_28_x86_64.whl (4543.9 MB)
```

```
PYTHONPATH=$wd/pip-installs \
```

```
srunc --jobid=$jobid -n1 --gpus 8 \
```

```
python3 -c 'import torch; print("I have this many devices:", torch.cuda.device_count())'
```

Make the freshly installed PyTorch available to your Python™ runs

NOTE: better to setup a **module file**

```
> I have this many devices: 8
```

Should yield the number of GCDs in the node.

PyTorch wheel install – virtual environments

- Virtual environments are convenient to manage Python™ package installation in your user-space

Leverage the venv module to create the virtual environment

We are happy to leverage system's already installed packages

```
python3 -m venv --system-site-packages cray-python-virtualenv
```

```
source cray-python-virtualenv/bin/activate
```

Activate the environment. It will be leveraged by the install and run.

Install and run as before. No need to specify install location – the environment is doing it for you.

```
pip3 install --pre torch==2.7.0+rocm6.3 --index-url https://download.pytorch.org/whl/  
srun --jobid=$jobid -n1 --gpus 8 \
```

```
python3 -c 'import torch; print("I have this many devices:", torch.cuda.device_count())'
```

PyTorch wheel install – conda environment

- Conda environment adds the package-manager functionality to a virtual environment
- One can tune the Python version to use as we won't be leveraging the system one anymore.

Download and install a minimal conda (miniconda) specific version.

```
curl -LO https://repo.anaconda.com/miniconda/Miniconda3-py310_24.1.2-0-Linux-x86_64.sh
```

Download and install a minimal conda (miniconda) latest version.

```
curl -LO https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86.sh
```

```
bash ./Miniconda3-* -b -p miniconda3 -s
```

Activate the conda environment

```
source $wd/miniconda3/bin/activate base
```

Create and activate a conda environment to install PyTorch based on Python 3.10

```
conda create -y -n pytorch python=3.10
```

```
source $wd/miniconda3/bin/activate pytorch
```

Install and run as before - Conda package manager doesn't have ROCm-enabled PyTorch installs

```
pip3 install --pre torch==2.7.0+rocm6.3 --index-url https://download.pytorch.org/whl/
```

```
srun --jobid=$jobid -n1 --gpus 8 \
```

```
python3 -c 'import torch; print("I have this many devices:", torch.cuda.device_count())'
```

TensorFlow wheel install

- Requirements similar to PyTorch install.
- Unlike PyTorch, TensorFlow packages for ROCm have a specific name:
 - `tensorflow-rocm`

Check the available versions from <https://pypi.org/project/tensorflow-rocm/#history>

Package version example:

- TensorFlow version 2.14.0
- Built on top of ROCm 6.0.0

tensorflow-rocm 2.14.0.600

pip install tensorflow-rocm

Released: Jan 10, 2024

TensorFlow is an open source machine learning framework for everyone.

Navigation

- Project description
- Release history
- Download files

Verified details

Maintainers

- JasonF-amd
- rsanthanam-amd
- sunway513

Release history

THIS VERSION	2.14.0.600	Jan 10, 2024
	2.13.1.600	Jan 10, 2024
	2.13.0.570	Oct 4, 2023
	2.12.1.600	Jan 10, 2024

TensorFlow wheel install

- Installing TensorFlow 2.14.0 for ROCm 6.0.0:

```
pip3 install tensorflow-rocm==2.14.0.600
```

- One can leverage pip command to get the available versions

```
pip3 install tensorflow-rocm==
```

```
ERROR: Could not find a version that satisfies the requirement tensorflow-rocm== (from versions: 2.8.0, 2.8.1, 2.8.2, 2.8.3, 2.8.4, 2.9.1, 2.9.2, 2.9.3, 2.9.4, 2.10.0.520, 2.10.0.530, 2.10.1.540, 2.10.1.550, 2.11.0.530, 2.11.0.540, 2.11.1.550, 2.12.0.560, 2.12.1.570, 2.12.1.600, 2.13.0.570, 2.13.1.600, 2.14.0.600)
ERROR: No matching distribution found for tensorflow-rocm==
```

- Checking number of GPUs:

```
python3 -c 'from tensorflow.python.client import device_lib ; device_lib.list_local_devices()'
```

```
Created device /device:GPU:0 with 63922 MB memory: -> device: 0, name: AMD Instinct MI210, pci bus id: 0000:63:00.0
```

```
Created device /device:GPU:1 with 63922 MB memory: -> device: 1, name: AMD Instinct MI210, pci bus id: 0000:43:00.0
```

```
Created device /device:GPU:2 with 63922 MB memory: -> device: 2, name: AMD Instinct MI210, pci bus id: 0000:03:00.0
```

```
Created device /device:GPU:3 with 63922 MB memory: -> device: 3, name: AMD Instinct MI210, pci bus id: 0000:26:00.0
```

JAX wheel install

- JAX GPU support comes from the package `jaxlib`
- `jaxlib` requires building from source upstream
- You can leverage the wheel files from ROCm Github:
 - <https://github.com/ROCm/jax/releases>

rocm-jax-v0.5.0 Latest

ROCm JAX 0.5.0 Release

ROCm 6.0.3, 6.2.4 and 6.3.1

Python 3.10.16

jaxlib

```
python3 -m pip install https://github.com/ROCm/jax/releases/download/rocm-jax-v0.5.0/jaxlib-0.5.0-cp310-cp310-abi3-linux_x86_64.whl
```

JAX ROCm Plugin

```
python3 -m pip install https://github.com/ROCm/jax/releases/download/rocm-jax-v0.5.0/jax_rocm60_pjrt-0.5.0-cp310-cp310-abi3-linux_x86_64.whl
```

Python 3.11.11

jaxlib

```
python3 -m pip install https://github.com/ROCm/jax/releases/download/rocm-jax-v0.5.0/jaxlib-0.5.0-cp311-cp311-abi3-linux_x86_64.whl
```

JAX ROCm Plugin

```
python3 -m pip install https://github.com/ROCm/jax/releases/download/rocm-jax-v0.5.0/jax_rocm60_pjrt-0.5.0-cp311-cp311-abi3-linux_x86_64.whl
```

JAX wheel install for Python™ 3.10

- Example installing commands for JAX version 0.5.0 :

- jaxlib

```
python3 -m pip install \
https://github.com/ROCm/jax/releases/download/rocm-jax-v0.5.0/jaxlib-0.5.0-cp310-cp310-manylinux\_2\_28\_x86\_64.whl
```

JAX package doesn't have any GPU
software dependency

- JAX ROCm Plugin

```
python3 -m pip install \
https://github.com/ROCm/jax/releases/download/rocm-jax-v0.5.0/jax\_rocm60\_plugin-0.5.0-cp310-cp310-manylinux\_2\_28\_x86\_64.whl
```

- JAX

```
python3 -m pip install \
https://github.com/ROCm/jax/archive/refs/tags/rocm-jax-v0.5.0.tar.gz
```

- Checking number of GPUs:

```
python -c 'import jax ; print("I have this many GPUs:", jax.local_device_count())'
```

- Should yield depending on the system:

```
I have this many GPUs: 8
```

BUILD FROM SOURCE

CHECK OUR HPCTRainingDOCK REPO

HPCTrainingDock repo: <https://github.com/amd/HPCTrainingDock>

Builds from source for:

- PyTorch:
https://github.com/amd/HPCTrainingDock/blob/main/extras/scripts/pytorch_setup.sh
- JAX:
https://github.com/amd/HPCTrainingDock/blob/main/extras/scripts/jax_setup.sh
- CuPy:
https://github.com/amd/HPCTrainingDock/blob/main/extras/scripts/cupy_setup.sh
- MPI4Py:
https://github.com/amd/HPCTrainingDock/blob/main/comm/scripts/mpi4py_setup.sh

TensorFlow source install

(tested with ROCm 6.4.0)

Clone desired version of TensorFlow

```
git clone --recursive -b merge-250318 https://github.com/ROCm/tensorflow-upstream
```

Install system and Python™ requirements

```
apt-get install python3-dev python3-pip openjdk-8-jdk openjdk-8-jre unzip wget git python-is-python3 patchelf
pip3 install numpy wheel mock future pyyaml setuptools requests keras_preprocessing keras_applications jupyter
```

← Might need sudo

Download Bazelisk: <https://github.com/bazelbuild/bazelisk/blob/master/README.md> and put it in your PATH:

```
curl -Lo bazelisk https://github.com/bazelbuild/bazelisk/releases/latest/download/bazelisk-$(uname -s | tr '[:upper:]' '[:lower:]')-amd64
chmod +x bazelisk && sudo mv bazelisk /usr/local/bin/bazel
```

← Can also move it to a different dir as long as it is in your PATH

Set USE_BAZEL_VERSION env variable to what is needed by TensorFlow

```
export USE_BAZEL_VERSION=$(cat tensorflow-upstream/.bazelversion | head -n 1)
```

Load necessary modules and set env variables

```
module load rocm amdclang
```

← Module definitions at:

https://github.com/amd/HPCTrainingDock/blob/main/rocm/scripts/rocm_setup.sh

If this command fails, you might need to manually edit the clang path in the .bazelrc file located in the tensorflow-upstream repo dir

Configure TensorFlow

```
cd tensorflow-upstream
```

```
yes "" | TF_NEED_CLANG=1 ROCM_PATH=$ROCM_PATH TF_NEED_ROCM=1 PYTHON_BIN_PATH=/usr/bin/python3 ./configure
```

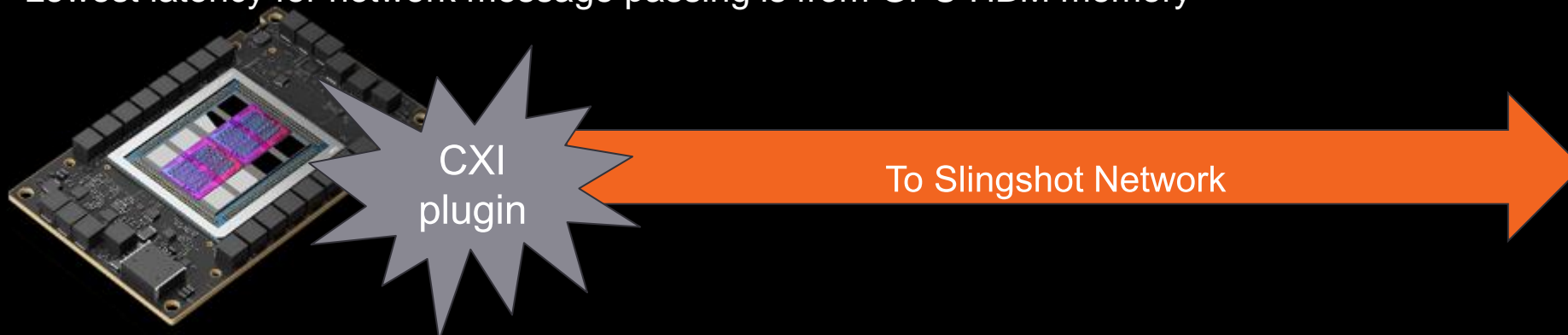
Install TensorFlow

```
bazel build --config=opt --config=rocm --repo_env=WHEEL_NAME=tensorflow_rocm --action_env=project_name=tensorflow_rocm/
//tensorflow/tools/pip_package:wheel --verbose_failures --repo_env=CC=$(which clang) --repo_env=BAZEL_COMPILER=$(which clang)
--repo_env=CLANG_COMPILER_PATH=$(which clang) && pip3 install --upgrade bazel-bin/tensorflow/tools/pip_package/wheel_house/tensorflow*.whl
```

←

Comms are important! - RCCL AWS-CXI plugin

- Several AMD sites directly attach AMD Instinct™ Accelerators to the Cray Slingshot Network
 - Enable collectives computation on devices
 - Minimize the role of the CPU in the control path – expose more asynchronous computation opportunities
 - Lowest latency for network message passing is from GPU HBM memory



- CXI plugin is a runtime dependency. Requires: HPE Cray libfabric implementation
 - <https://github.com/ROCm/aws-ofi-rccl>
 - 3-4x faster collectives

- **CXI plugin is NOT featured in official DockerHub images! Make sure it exists in your environment!**

```
export NCCL_DEBUG=INFO
export NCCL_DEBUG_SUBSYS=INIT
# and search the logs for:
[0] NCCL INFO NET/OFI Using aws-ofi-rccl 1.4.0
```

Horovod install

- Horovod is a framework to enable distributed deep-learning training with TensorFlow, Keras, PyTorch, and Apache MXNet. The goal of Horovod is to make distributed deep learning fast and easy to use.

```
# Configure for ROCm
export HOROVOD_WITHOUT_MXNET=1
export HOROVOD_WITHOUT_GLOO=1
export HOROVOD_GPU=ROCM
export HOROVOD_ROCM_HOME=$ROCM_PATH
export HOROVOD_GPU_OPERATIONS=NCCL
export HOROVOD_CPU_OPERATIONS=MPI
export HOROVOD_WITH_MPI=1
export HOROVOD_ROCM_PATH=$ROCM_PATH
export HOROVOD_RCCL_HOME=$ROCM_PATH/rccl
export RCCL_INCLUDE_DIRS=$ROCM_PATH/rccl/include
export HOROVOD_RCCL_LIB=$ROCM_PATH/rccl/lib
export HCC_AMDGPU_TARGET=gfx90a
export CMAKE_PREFIX_PATH=$MPICH_PATH
```

Step 1: configure ROCm details

Step 2: configure for your favorite framework details

```
# Configure for TensorFlow
export HOROVOD_WITH_TENSORFLOW=1
export HOROVOD_WITHOUT_PYTORCH=1
```

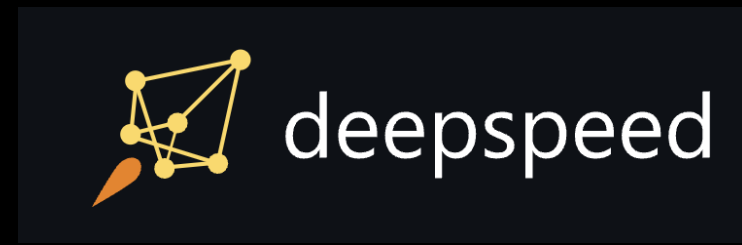
```
# Configure for PyTorch
export HOROVOD_WITHOUT_TENSORFLOW=1
export HOROVOD_WITH_PYTORCH=1
```

Horovod needs MPI at launch

```
# Install
pip install --no-cache-dir --force-reinstall --verbose horovod==$HOROVOD_VERSION
```

Step 3: install





DeepSpeed install

- DeepSpeed is a framework to optimize distributed deep-learning training and inference

```
DS_BUILD_AIO=0 \
DS_BUILD_CCL_COMM=1 \
DS_BUILD_CPU_ADAM=0 \
DS_BUILD_CPU_LION=0 \
DS_BUILD_EVOFORMER_ATTN=0 \
DS_BUILD_FUSED_ADAM=1 \
DS_BUILD_FUSED_LION=1 \
DS_BUILD_CPU_ADAGRAD=0 \
DS_BUILD_FUSED_LAMB=1 \
DS_BUILD_QUANTIZER=0 \
DS_BUILD_RANDOM_LTD=0 \
DS_BUILD_SPARSE_ATTN=0 \
DS_BUILD_TRANSFORMER=0 \
DS_BUILD_TRANSFORMER_INFERENCE=0 \
DS_BUILD_STOCHASTIC_TRANSFORMER=1 \
pip install deepspeed==0.14.0 \
--global-option="build_ext" --global-option="-j32"
```

Select all the optimizations
not all are enabled for GPUs

Allow multiple process builds

ds_report

Utility to report supported capabilities

```
-----
op name ..... installed .. compatible
-----
async_io ..... [NO] ..... [NO]
fused_adam ..... [YES] ..... [OKAY]
cpu_adam ..... [NO] ..... [OKAY]
cpu_adagrad ..... [NO] ..... [OKAY]
cpu_lion ..... [NO] ..... [OKAY]
evoformer_attn ..... [NO] ..... [NO]
fused_lamb ..... [YES] ..... [OKAY]
fused_lion ..... [YES] ..... [OKAY]
inference_core_ops ..... [NO] ..... [OKAY]
cutlass_ops ..... [NO] ..... [OKAY]
transformer_inference .. [NO] ..... [OKAY]
quantizer ..... [NO] ..... [OKAY]
ragged_device_ops ..... [NO] ..... [OKAY]
ragged_ops ..... [NO] ..... [OKAY]
random_ltd ..... [NO] ..... [OKAY]
sparse_attn ..... [NO] ..... [NO]
spatial_inference ..... [NO] ..... [OKAY]
transformer ..... [NO] ..... [OKAY]
stochastic_transformer . [YES] ..... [OKAY]
-----
```

PyTorch example

- What provides distributed capability:
 - PyTorch Distribute Data Parallel (DDP) – Batches of different data run concurrently
 - Other more sophisticated methods available
 - Frameworks like Deepspeed and Horovod can also enable distributed training.

```
import torch.distributed as dist
```

```
...
```

```
dist.init_process_group(
```

Let's use RCCL
collectives library

```
backend='nccl',
```

```
init_method='env://',
```

We'll be learning about the distributed setting
from the environment

```
world_size=int(os.environ['WORLD_SIZE']),
```

```
rank=int(os.environ['RANK']))
```

Capture some env vars to adjust my distributed
training

```
... Epoch 0 Loss 0.148397 Global batch size 2048 on 16 ranks
```

```
... Epoch 1 Loss 0.147906 Global batch size 2048 on 16 ranks
```

```
... Epoch 2 Loss 0.147717 Global batch size 2048 on 16 ranks
```

Frameworks tend to
include information on how
the training is distributed

TensorFlow + Horovod example

- What provides distributed capability: Horovod wraps the TensorFlow operators

```
import tensorflow as tf
import horovod.tensorflow as hvd
```

← Instantiate Horovod implementation for TensorFlow

```
# Horovod: initialize Horovod.
hvd.init()
```

← Horovod requires an initialization similar to what MPI requires: it will leverage MPI dependency to record the information about the ranks

```
# Horovod: pin GPU to be used to process local rank (one GPU per process)
tf.config.experimental.set_visible_devices(gpus[hvd.local_rank()], 'GPU')
```

← Horovod provides information about local rank useful for GPU binding

```
# A local optimizer.
opt = tf.optimizers.SGD(0.01)
```

```
# Wrap the Tensorflow operator (e.g. a gradient tape) with the corresponding
# Horovod distributed operator.
tape = tf.GradientTape()
tape = hvd.DistributedGradientTape(tape, compression=compression)
```

```
# Apply the gradient information gather from the distributed operator into the local optimizer.
gradients = tape.gradient(loss, model.trainable_variables)
opt.apply_gradients(zip(gradients, model.trainable_variables))
```

← Leverage the results of the distributed operator in the local optimizer

DISCLAIMERS AND ATTRIBUTIONS

The information contained herein is for informational purposes only and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD's products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale. GD-18

THIS INFORMATION IS PROVIDED 'AS IS.' AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

© 2025 Advanced Micro Devices, Inc. All rights reserved.

AMD, the AMD Arrow logo, Radeon™, Instinct™, EPYC, Infinity Fabric, ROCm™, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

The OpenMP name and the OpenMP logo are registered trademarks of the OpenMP Architecture Review Board

Windows is a registered trademark of Microsoft Corporation in the US and/or other countries.

Git and the Git logo are either registered trademarks or trademarks of Software Freedom Conservancy, Inc., corporate home of the Git Project, in the United States and/or other countries

Intel is a trademark of Intel Corporation or its subsidiaries

AMD 