Is Your AI Data Pre-processing Fast Enough? Speed It Up Using rocAL™

Rajy Rawther
PMTS Software Architect
Advanced Micro Devices, Inc.

Agenda

- Introduction: Why do we need rocAL?
- rocAL pipeline and architecture
- Operators for data loading and augmentations
- Flexible pipelines: scalability across multiple devices
- How to use rocAL?
- Deep dive into MLPerf object detection example with rocAL
- rocAL performance advantages
- rocAL use case in inference
- Conclusion

The Problem

Many Data Formats

Lots of Frameworks

What processor to choose?

FileList
TFRecord
RecordIO
LMDB

PYTORCH
Caffe2
TensorFlow
mxnet
- - - - - -

GPU
CPU
Custom
FPGA

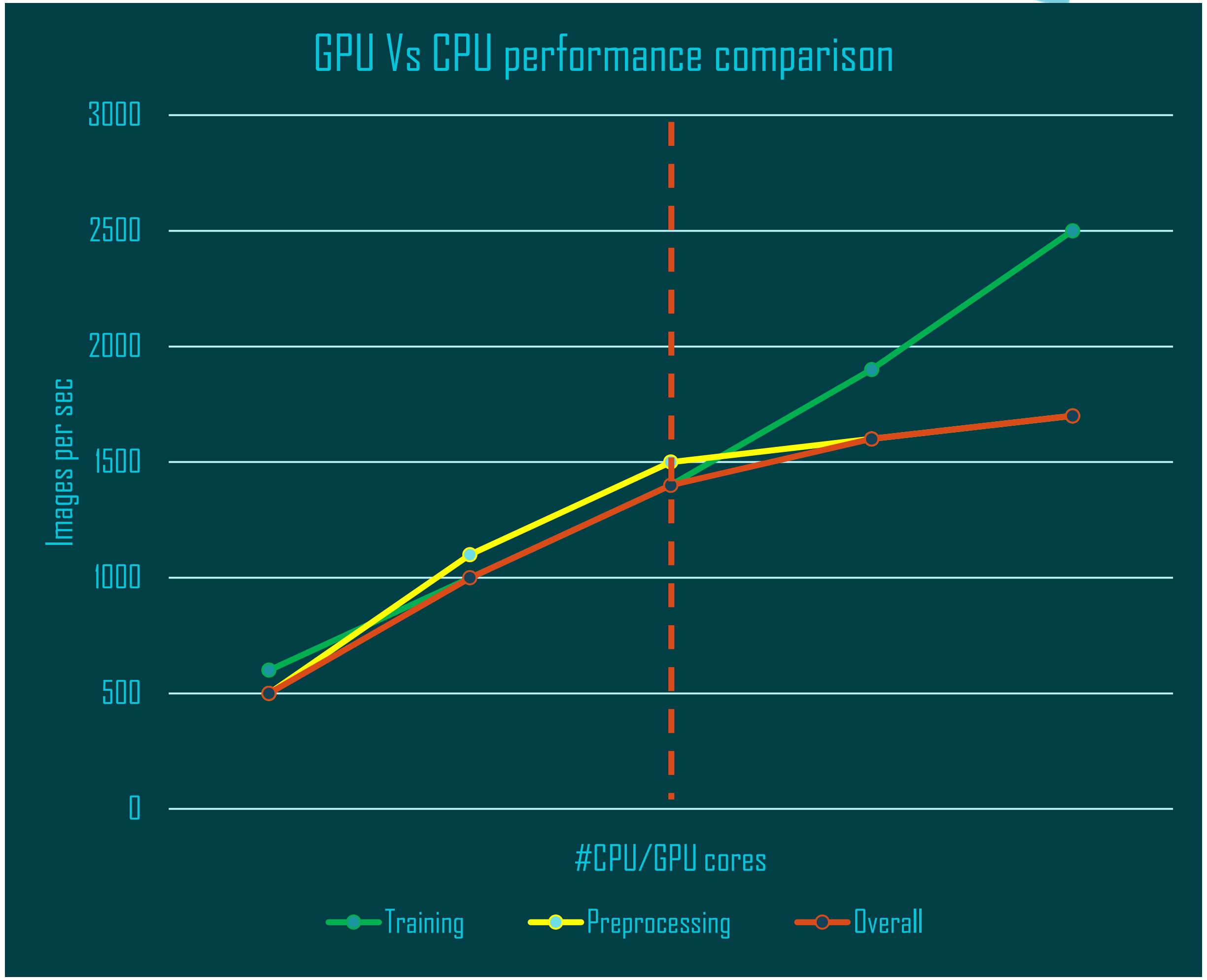
Needs a unified Library

Each Framework has its own data processing pipeline and each needs to be optimized individually

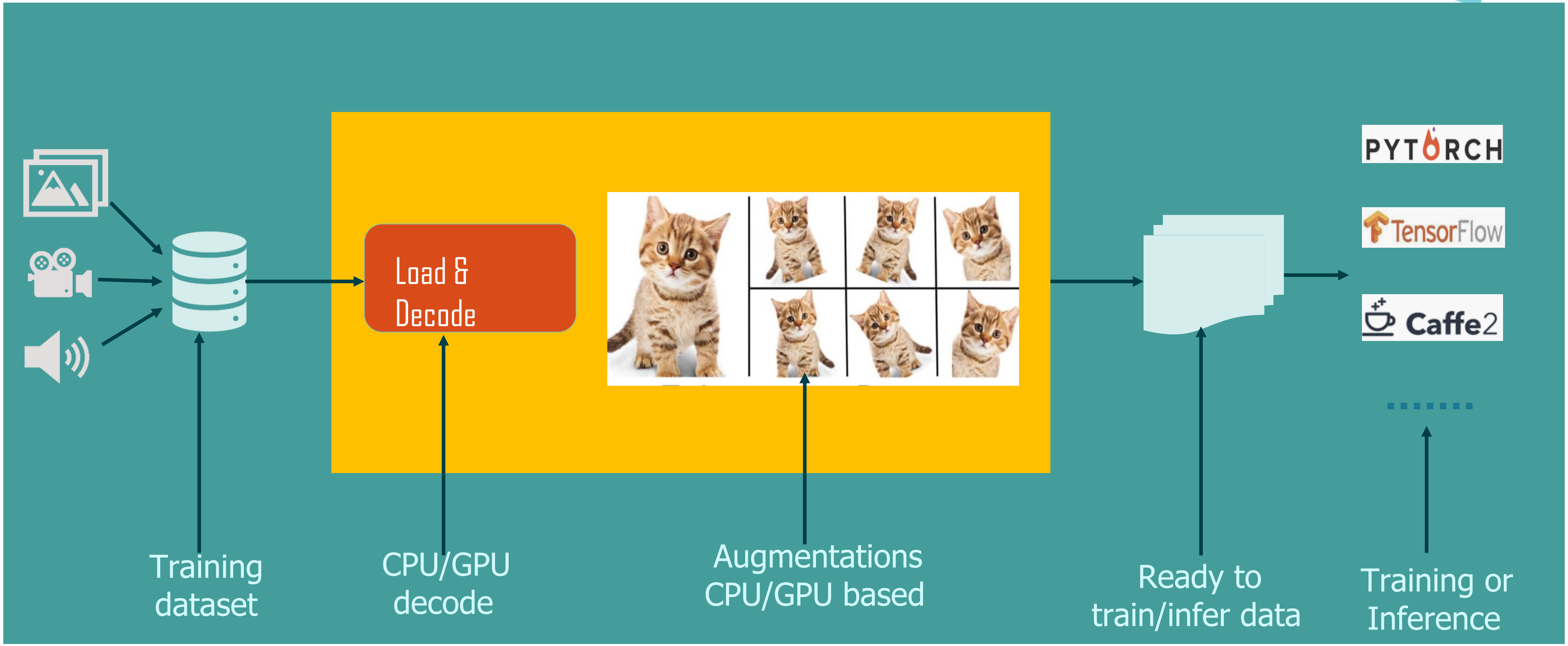
Feeding the Beast: How to Fully Utilize GPUs?

- GPU performance increases >2x every new generation
- Native pipelines mostly use CPU cores for pre-processing data before training
- As training gets faster with GPUs, the pre-processing needs to catch up
  - EPYC™ + MI100: 64 CPU cores, 8 GPUs, 8 CPU cores/GPU
  - EPYC™ + MI200: 64 cores, 8 GPUs(2x perf), 8 CPU cores/GPU, >300 TFLOPs (FP16)
- Falling CPU/GPU performance throttles the overall speed

High-performance pre-processing library which can load balance between CPU and GPU



Introducing ROCm Augmentation Library (rocAL™)

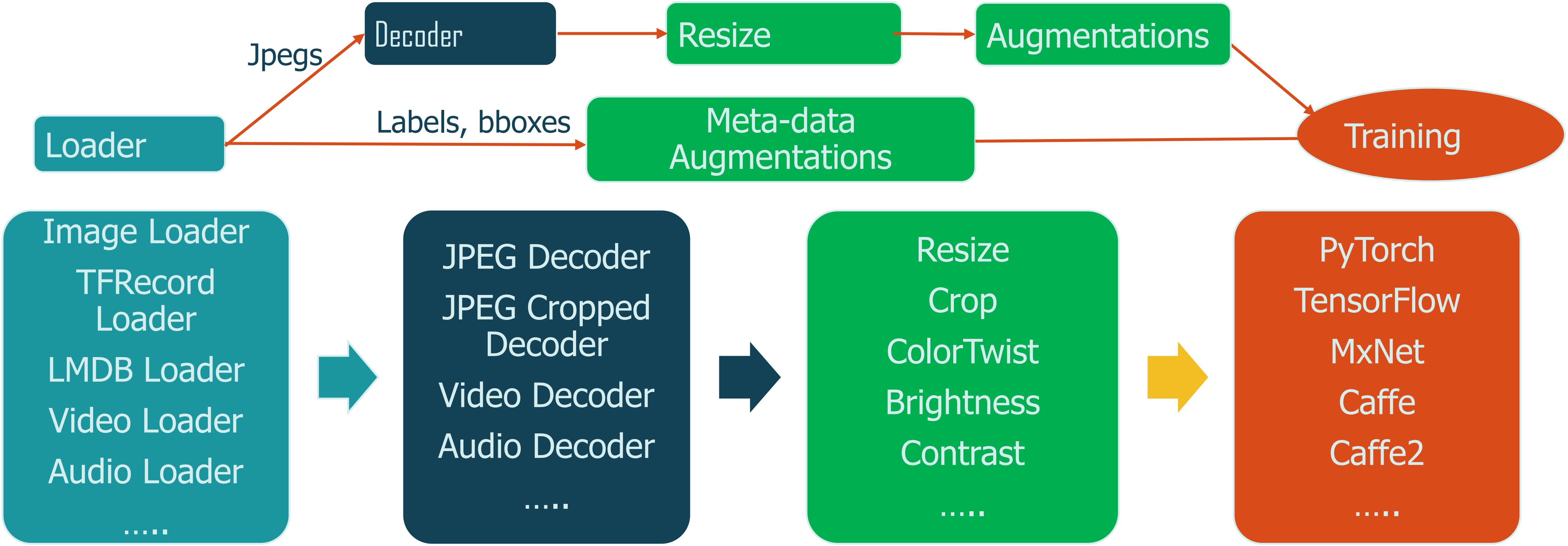


Key Features of rocAL

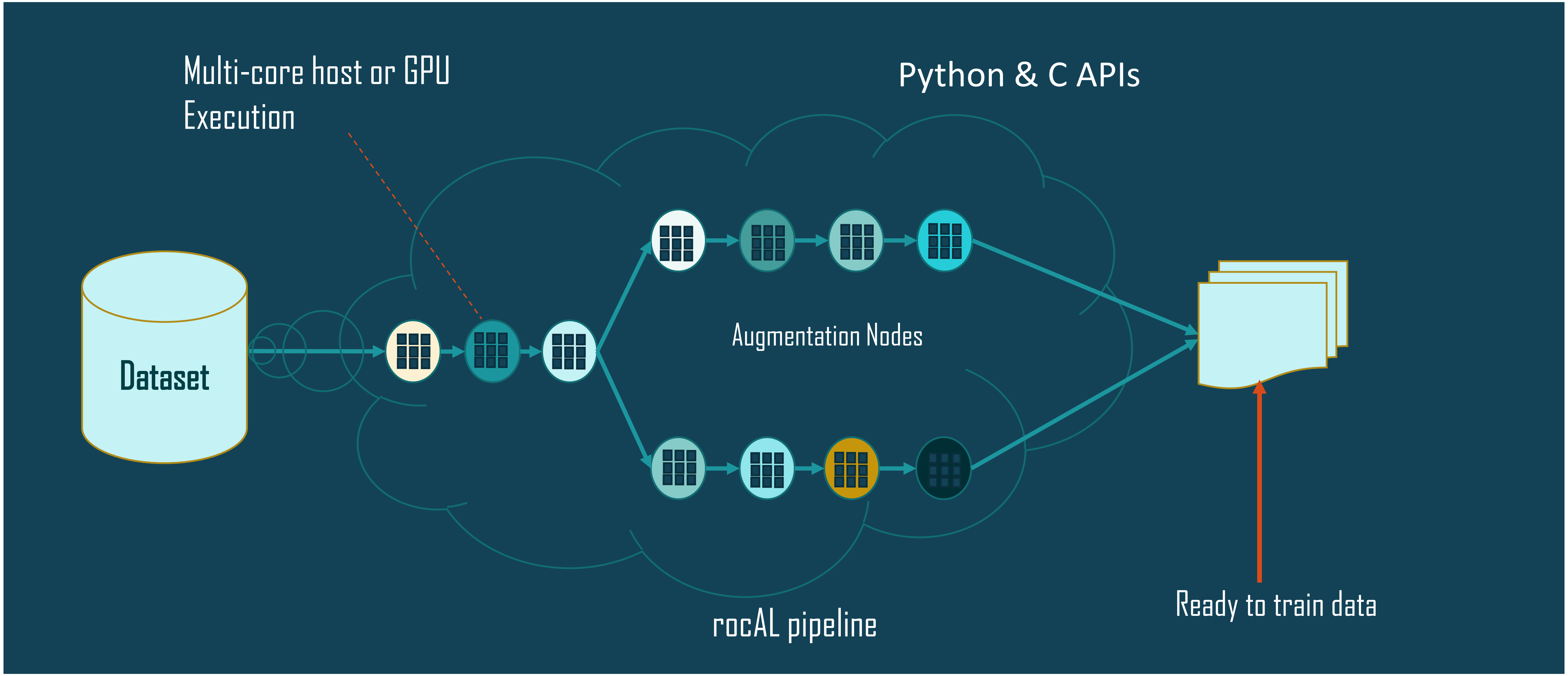
- CPU and GPU based implementations for each operators
- Python and C++ APIs for easy integration and testing
- Flexible graphs to create custom pipeline utilizing CPU cores or GPU
- Supports many new augmentations like fish-eye, non-linear blend, water, RICAP, etc.
- Support for many workloads
  - Classification
  - Object detection
  - Pose estimation and segmentation
- Seamless interoperability with frameworks using rocAL framework plugins
- Optimized to give maximum performance on AMD EPYC™ CPUs and AMD Instinct™ GPUs

What is a rocAL Pipeline?

A pipeline is a graph of data flow connected by node operators

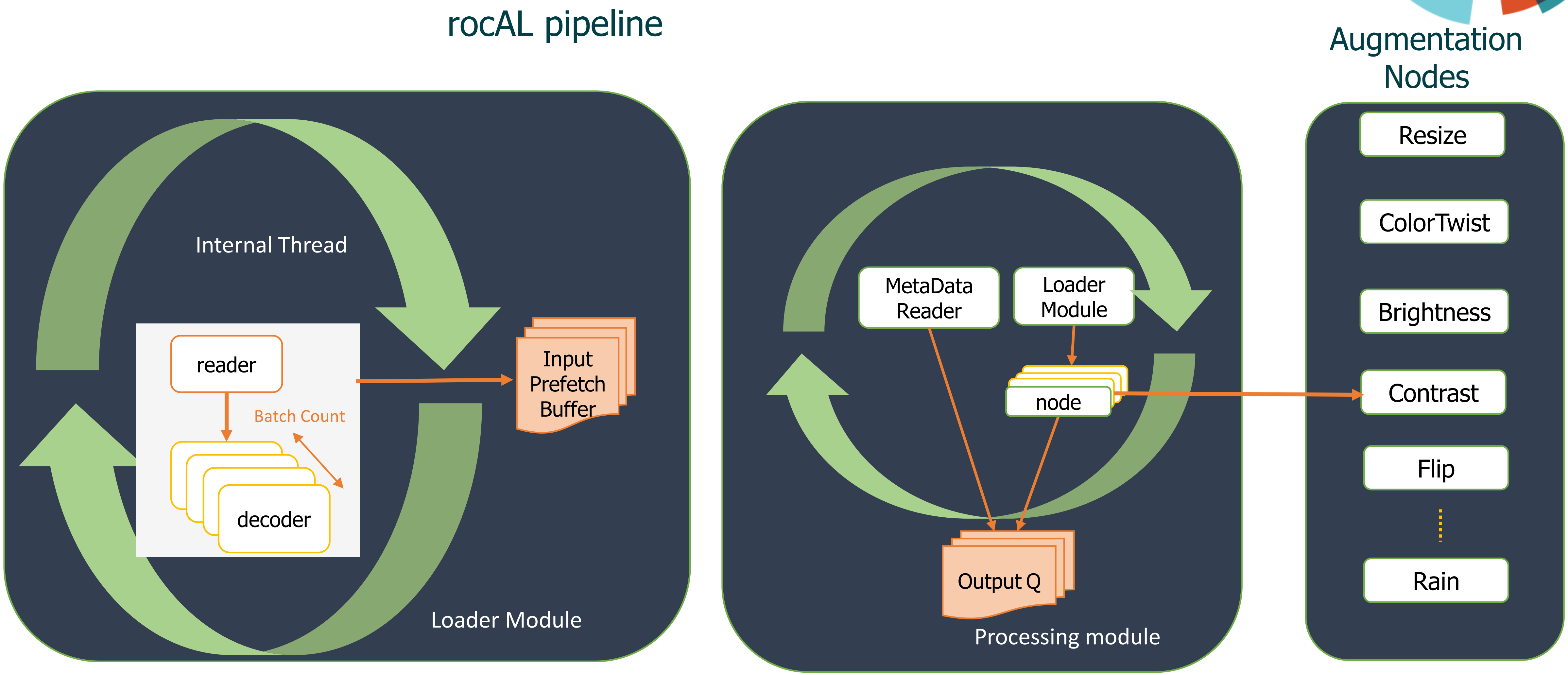


rocAL Pipeline Dataflow



AMD

rocAL Architecture

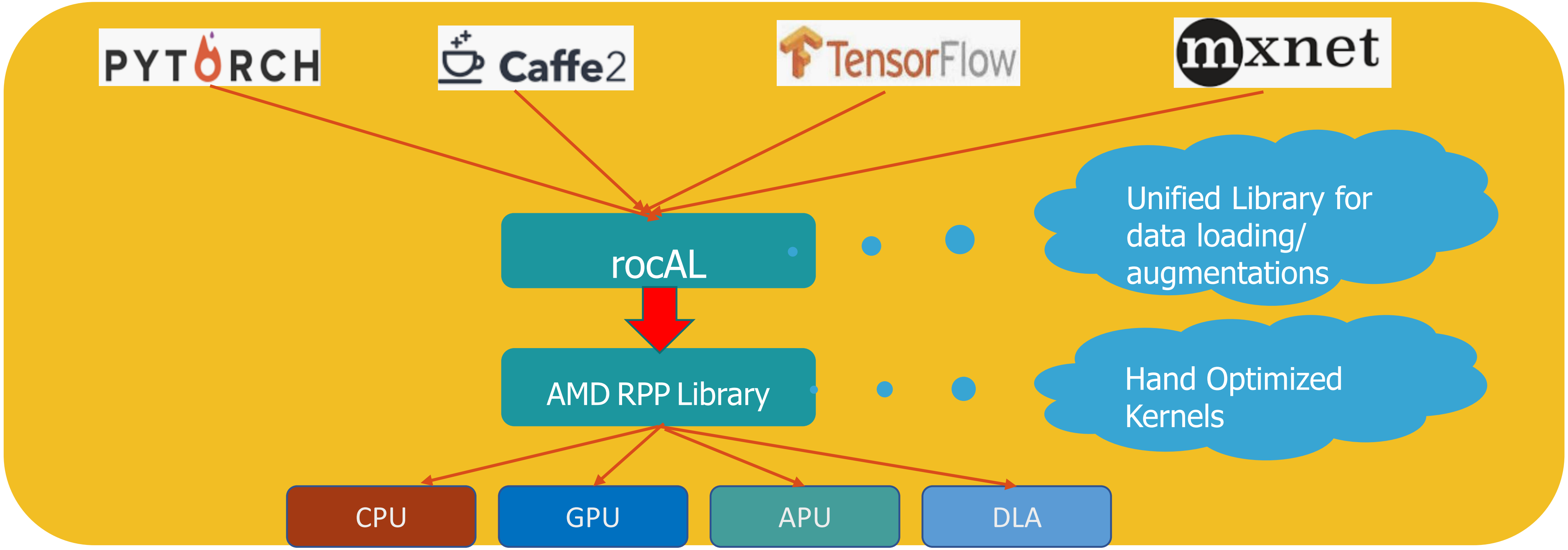


rocAL Operators

Reader	Loader	Decoder	ImageAug
File Reader	ImageLoader	Tjpeg Decoder	Resize
COCO Reader	ImageLoaderSharded	Cropped Decoder	Crop
TF Reader	Video Loader	HW Decoder	ColorTwist
RecordIO Reader	Audio Loader	Video Decoder	Normalize
LMDB Reader	SequenceLoader	Audio Decoder	Rotate
			Flip

rocAL Advantage

- One unified library that integrate to all the frameworks
- Optimized augmentation operations used among all
- Flexible to support different data formats (File folder reading, LMDB, TF Record, Record IO). Portable between frameworks



Using rocAL

- rocAL pipelines can be accessed using three simple steps (Define/Build and Run)

```
from amd.rocal.pipeline import pipeline_def
import amd.rocal.fn as fn

@pipeline_def
def example_pipeline():
    jpegs, labels = fn.readers.file(file_root=file_dir)
    images = fn.decoders.image(jpegs, device=decoder_device)
    resized_images = fn.resize(images, device, resize_w, resize_h)
    return resized_images, labels
```

Define

```
pipe = example_pipeline(batch_size=8, num_threads=32,device_id=0)
pipe.build()
```

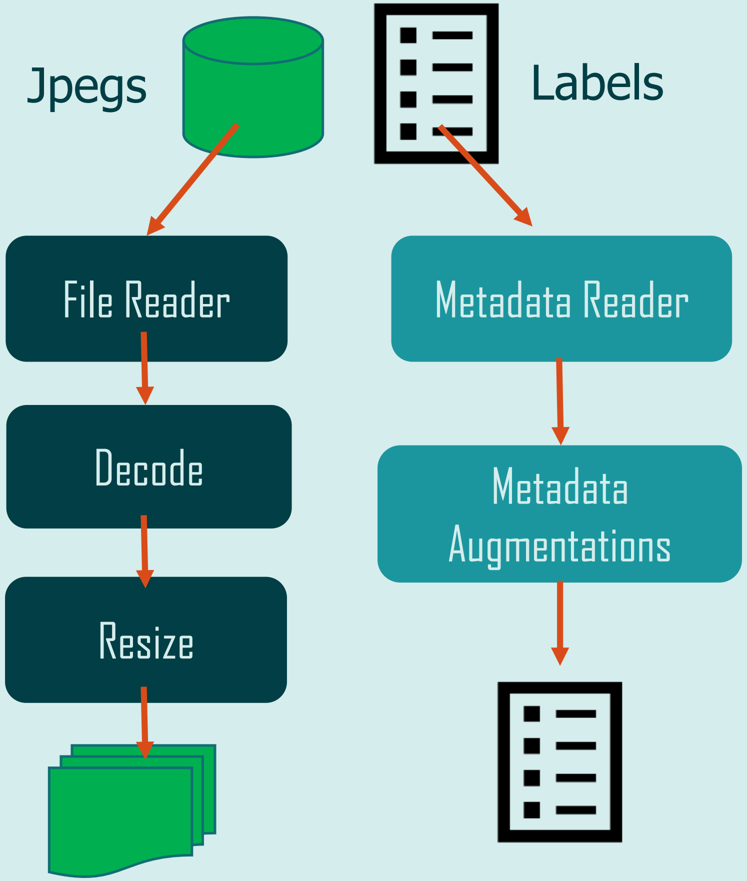
Build

```
images, labels = pipe.run()
```

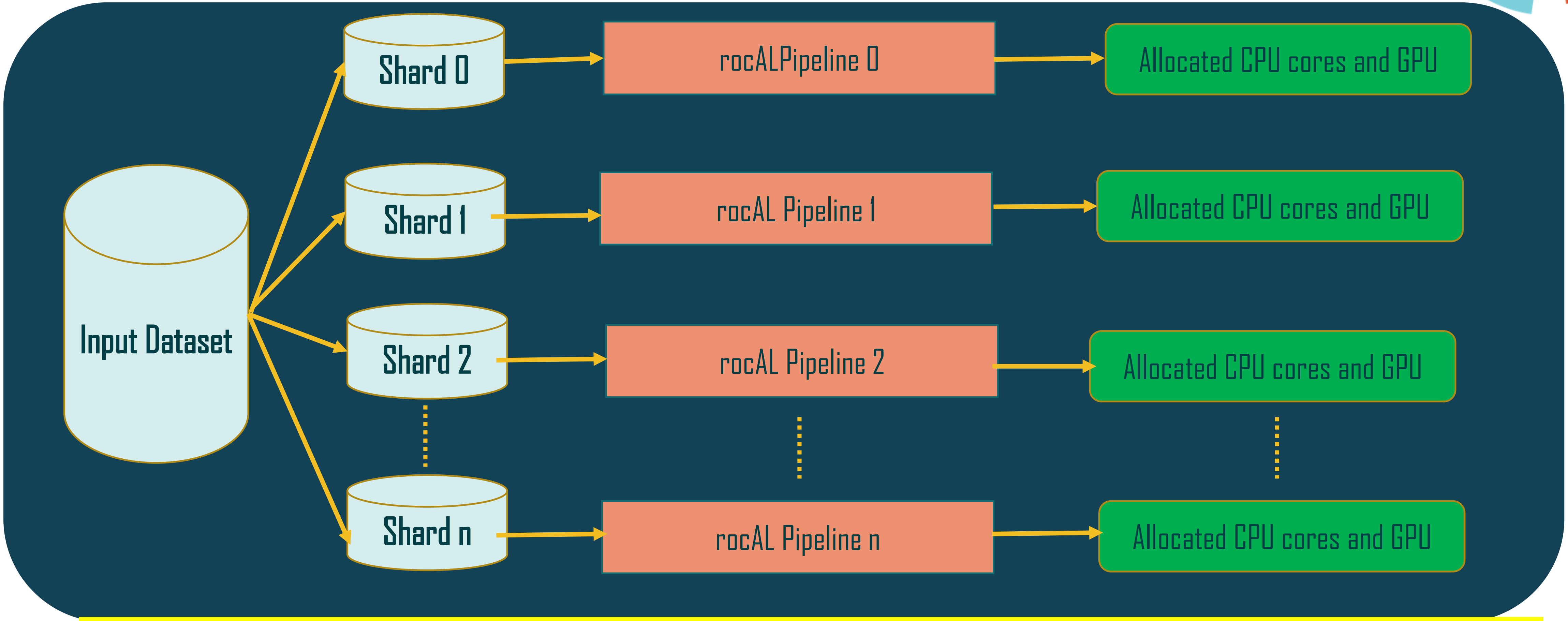
Run

Sample Output From Example_ Pipeline

Each sample is decoded and resized to 224x224

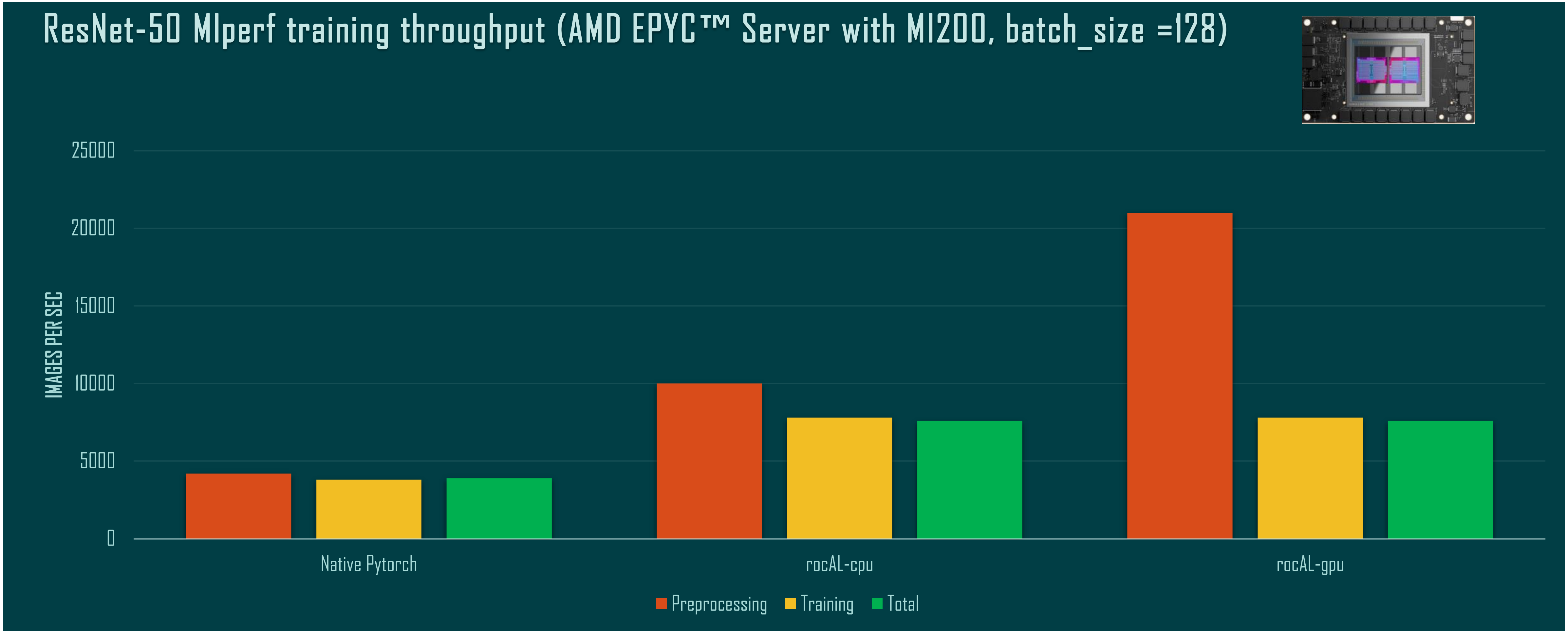


Flexible Pipelines: Scalability Across Devices



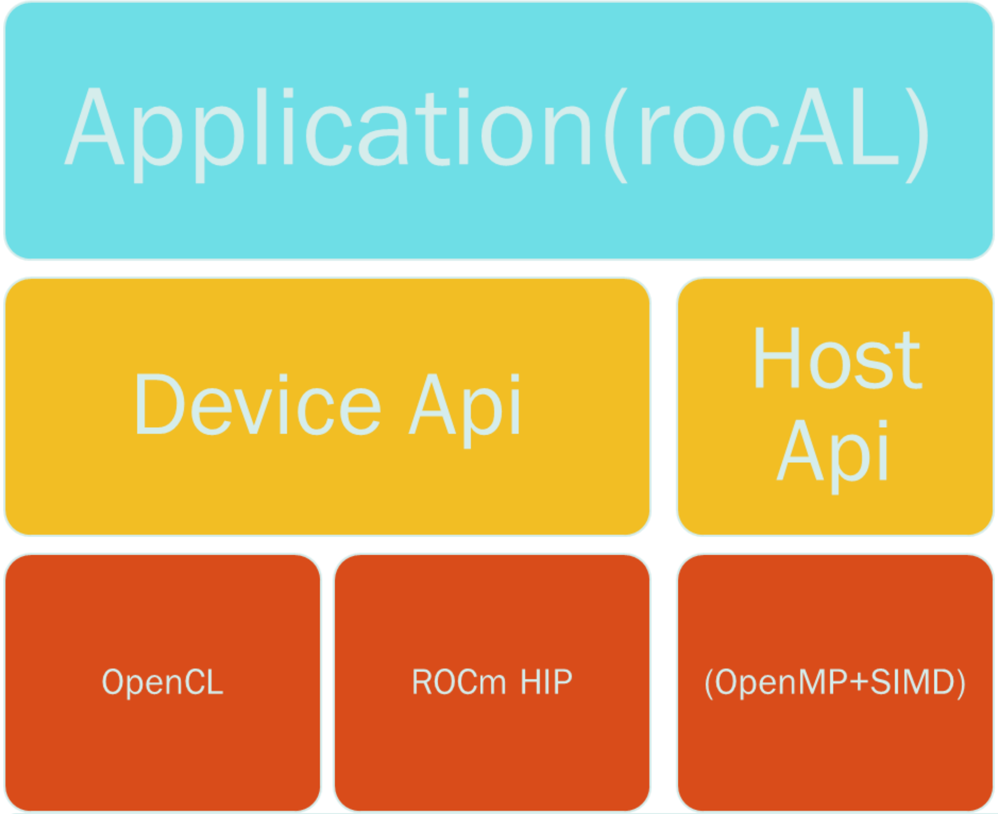
Each pipeline is configured with GPU device_id and CPU core bindings

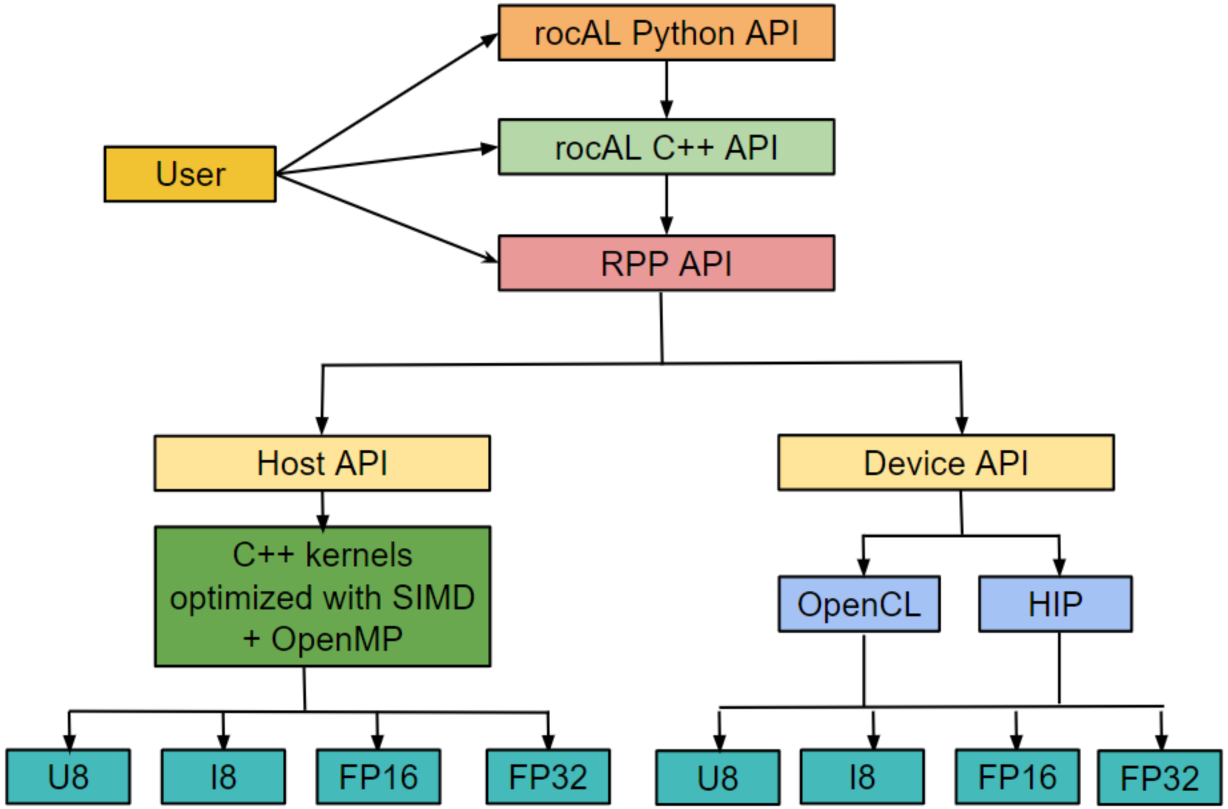
rocAL's Impact In Performance (Mlperf Resnet-50 Training)



rocAL's Core: AMD RPP Library

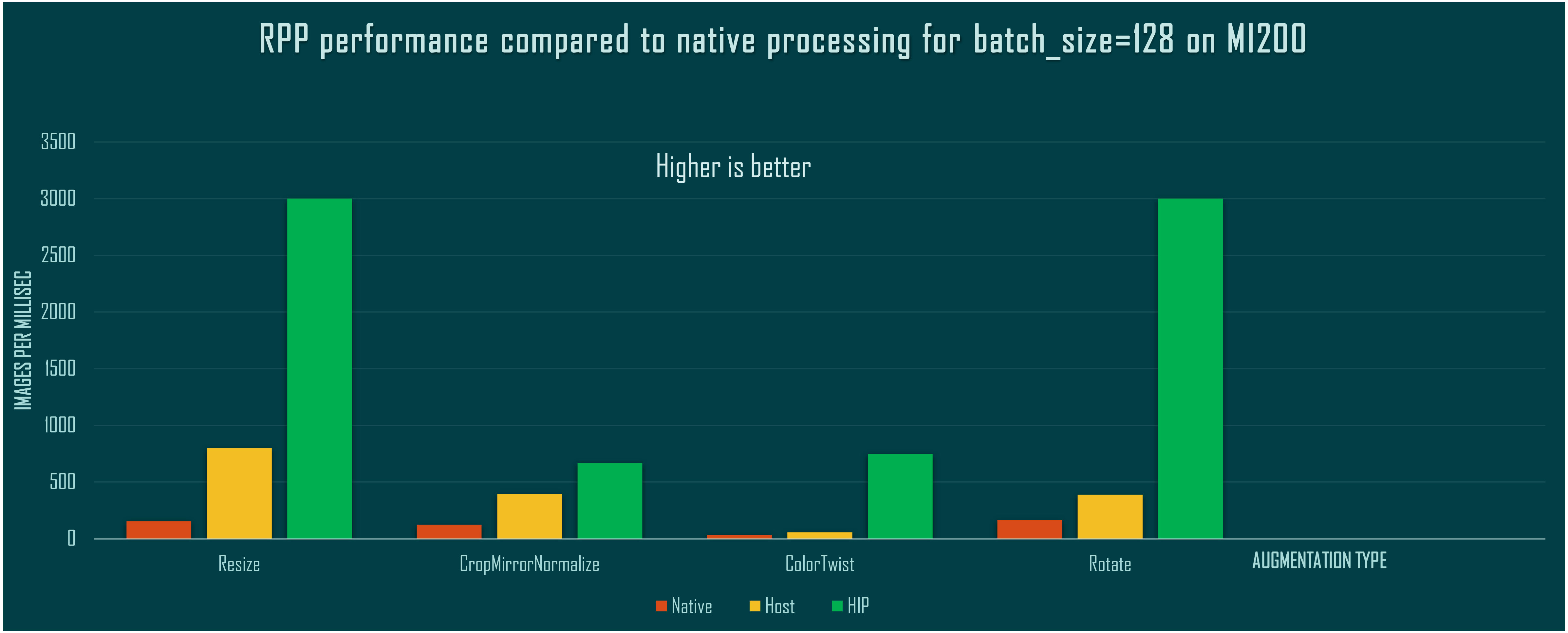
Radeon Performance Primitives (RPP) Library



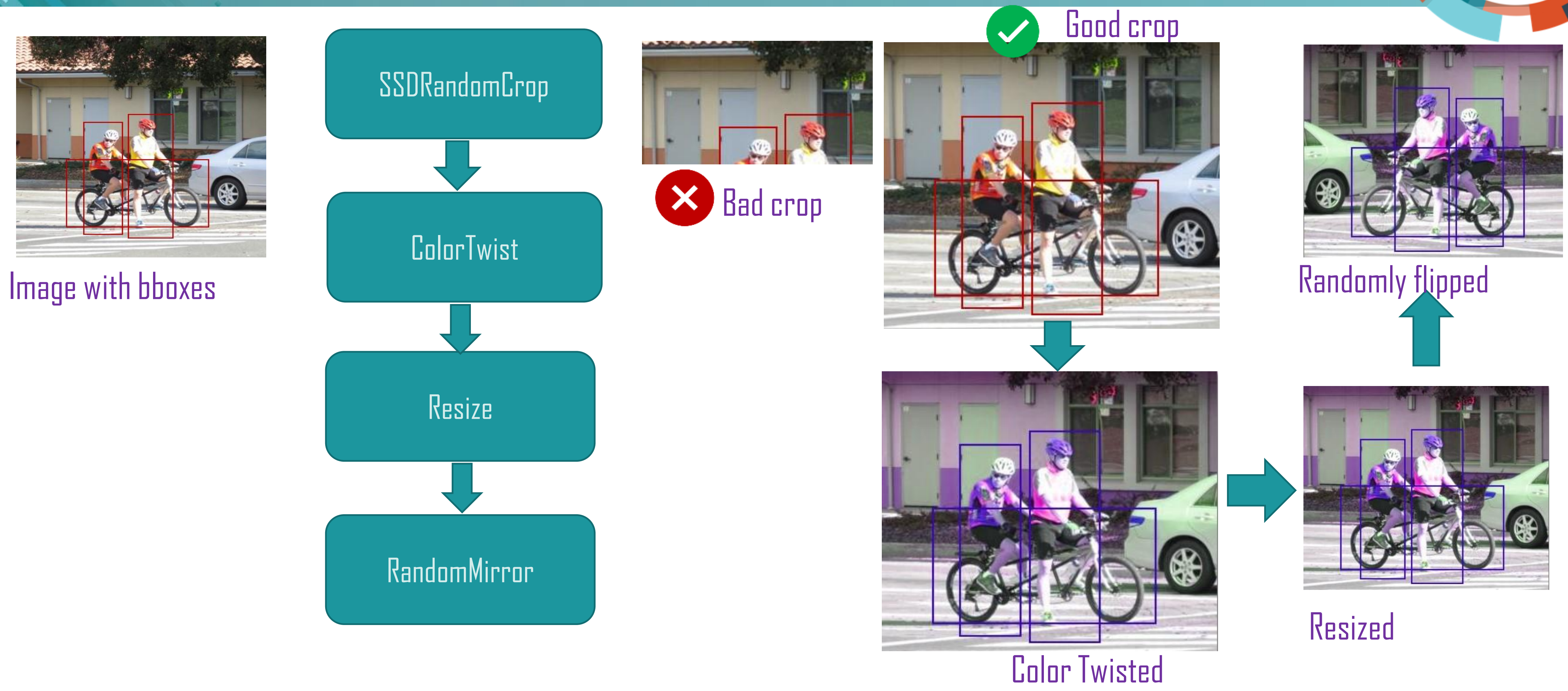


Hand Optimized High Performance Library Under AMD's ROCm Stack

RPP Advantage



Example: SSD Object Detection Training Augmentations



Mlperf SSD Training With rocAL

Define

```
def COCOPipeline( batch_size, num_threads, local_rank , world_size, device_id, data_dir, ann_dir):
    pipe = Pipeline(batch_size, num_threads, device_id=device_id)
    with pipe:
        jpegs, bboxes, labels = fn.readers.coco(path=data_dir, random_shuffle=True)
        crop_begin, crop_size, bboxes, labels = fn.random_bbox_crop(bboxes, labels, device="cpu",
                                                    aspect_ratio=[0.5, 2.0],
                                                    thresholds=[0, 0.1, 0.3, 0.5, 0.7, 0.9])
        images_decoded = fn.decoders.image_slice(jpegs, crop_begin, crop_size, device="cpu", type = types.RGB)
        res_images = fn.resize(images_decoded, device="gpu", resize_x=crop, resize_y=crop)
        cl_twist_images = fn.color_twist(res_images, device="gpu", contrast_rand, brightness_rand, hue_rand)
        bboxes = fn.bb_flip(bboxes, ltrb=True, horizontal=flip_coin)
        images = fn.crop_mirror_normalize(cl_twist_images, device="gpu",
                                          crop=(crop, crop),
                                          mirror=flip_coin,
                                          mean=[0.485*255,0.456*255 ,0.406*255 ],
                                          std=[0.229*255 ,0.224*255 ,0.225*255 ])
        bboxes, labels = fn.box_encoder(bboxes, labels, device=rali_device)
        pipe.set_outputs(images, bboxes, labels)
```

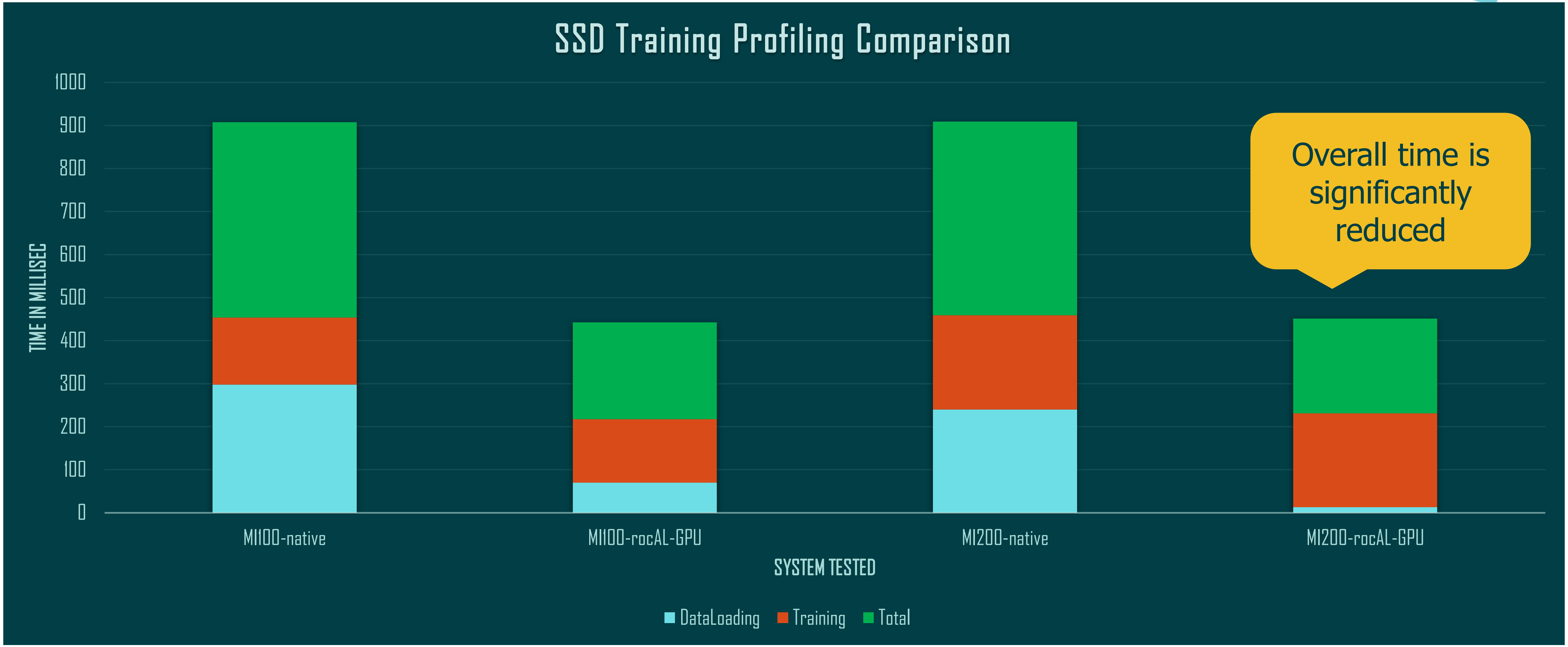
```
train_loader = rocALGenericIterator(pipe)
pipe.build()

for i, data in enumerate(train_loader):
  images, bboxes, labels = data
  # do model training
```

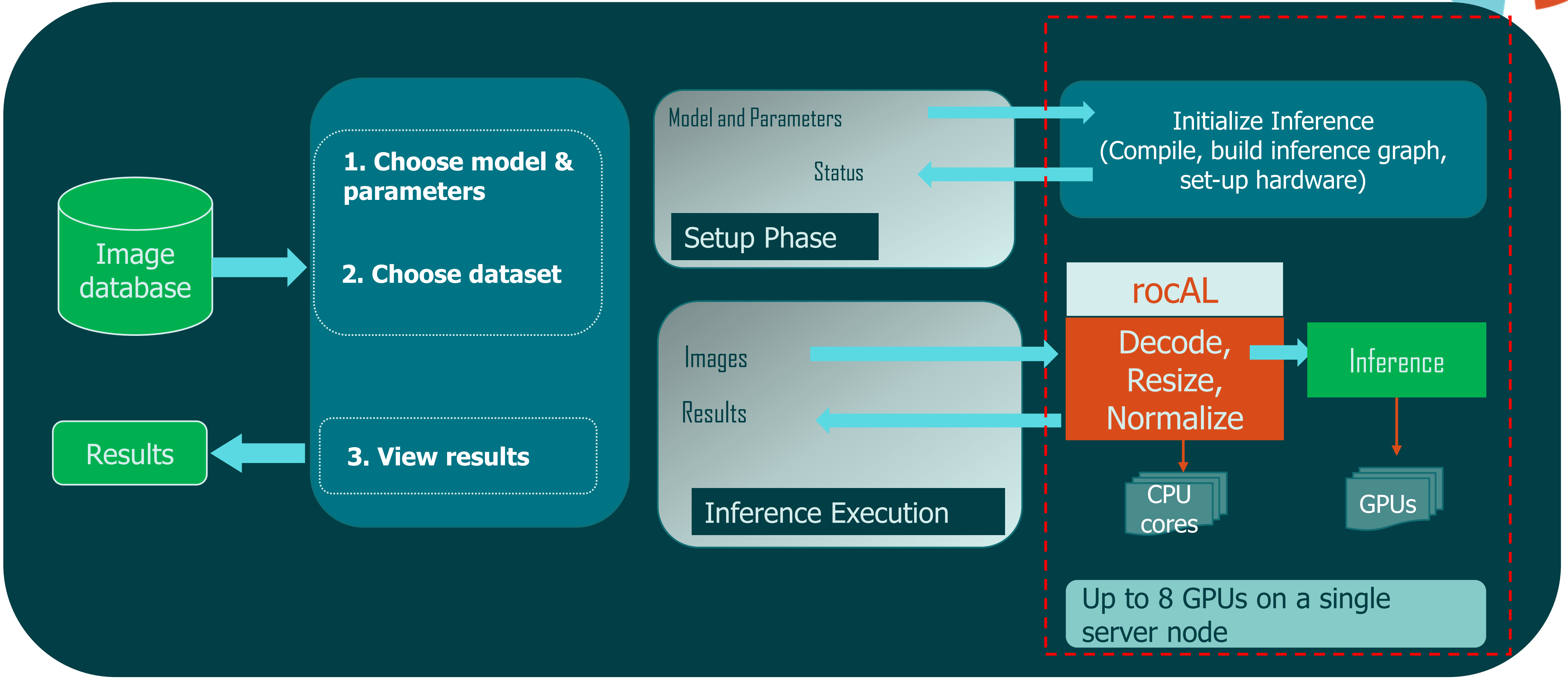
Build

Run

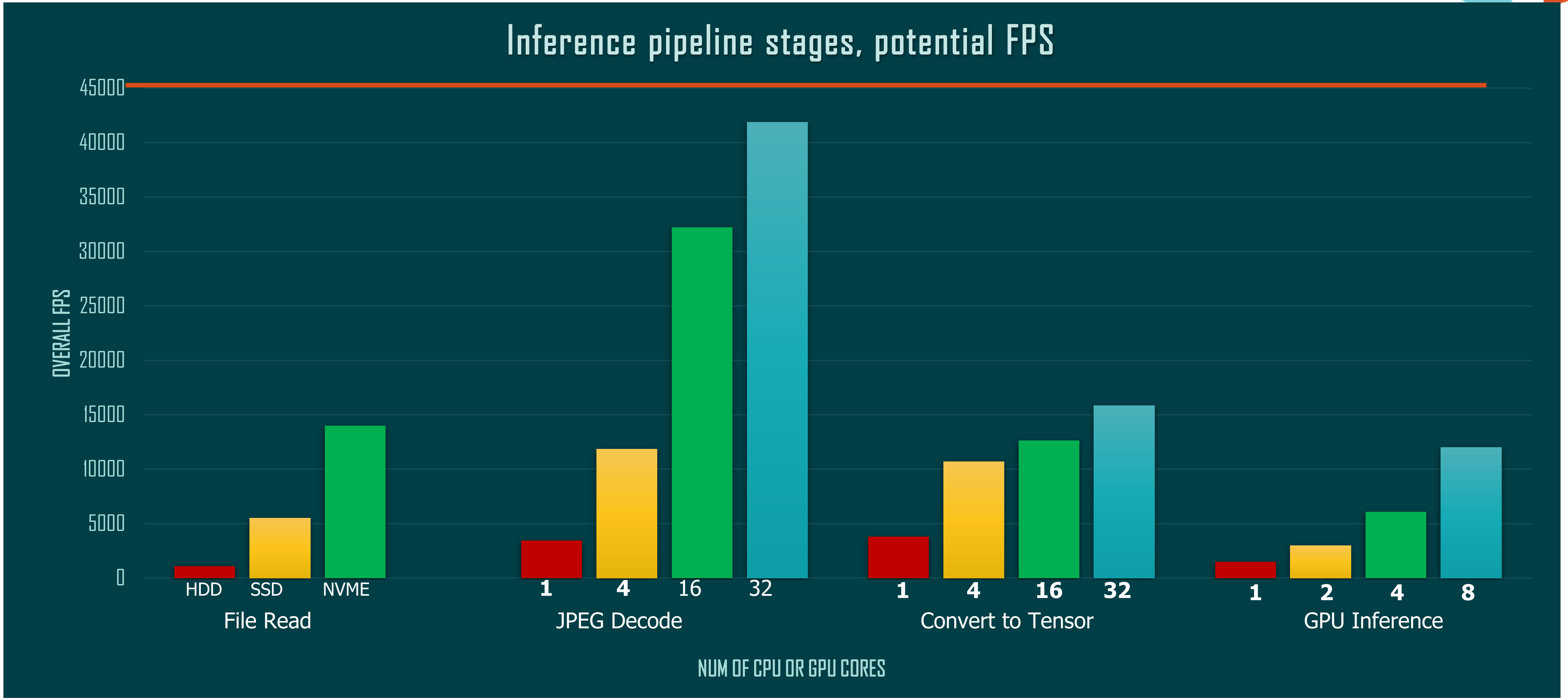
rocAL Advantage in MLPerf SSD Training



rocAL Use Case In Inference: Inference Server



Different Stages Of Inference Pipeline



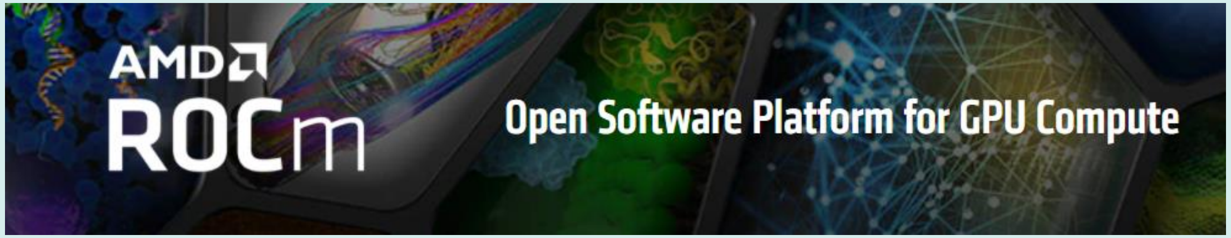
AMD

Challenges

- Meta data augmentations and new data-types are introduced to help with bounding-box and other meta-data augmentations
- CPU based decoding has a hit on performance even with TJpeg decoder
  - Hardware decoder using VCN
  - ROI based decoding
- Memory management is tricky when we use mixed devices and variable batch_size
- Discrepancies in image-processing transforms across different frameworks. Transforms produce different outputs.
- Video processing needs new data layout to represent sequences (NFHWC)

Conclusion

- rocAL is the AMD open source accelerated data augmentation and data loading library
- It provides full pre-processing pipelines to be used for training or inference
- Has easy framework integration for today's machine learning workloads
- rocAL's hybrid pipelines help intelligent load balancing between CPU and GPU
- It is portable across multiple framework with one underlying library
- The AMD Open sourced RPP library provides the backbone for rocAL



References

rocAL

https://github.com/GPUOpen-ProfessionalCompute-Libraries/MIVisionX/tree/master/rocAL

MIVisionX

https://gpuopen-professionalcompute-libraries.github.io/MIVisionX/

RPP

https://github.com/GPUOpen-ProfessionalCompute-Libraries/MIVisionX

AMD ROCm

https://rocmdocs.amd.com/en/latest/

Disclaimer

- The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.
- 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.
-
- AMD, the AMD Arrow logo, EPYC, Radeon, MI100, MI200, rocAL, RPP, MIVisionX, 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.

AMD