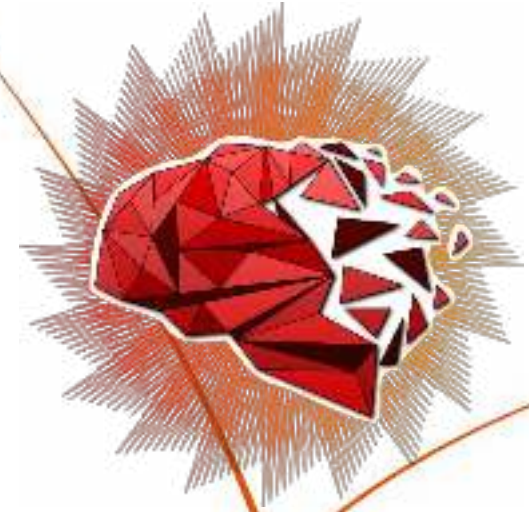


Orchestrating a brighter world

**NEC**



## **SOL: Transparent Neural Network Acceleration on NEC SX-Aurora TSUBASA**

**Dr. Nicolas Weber (NEC Labs Europe)**

# Where to start?

Where to start?



TensorFlow



Chainer

theano



Caffe2

mxnet

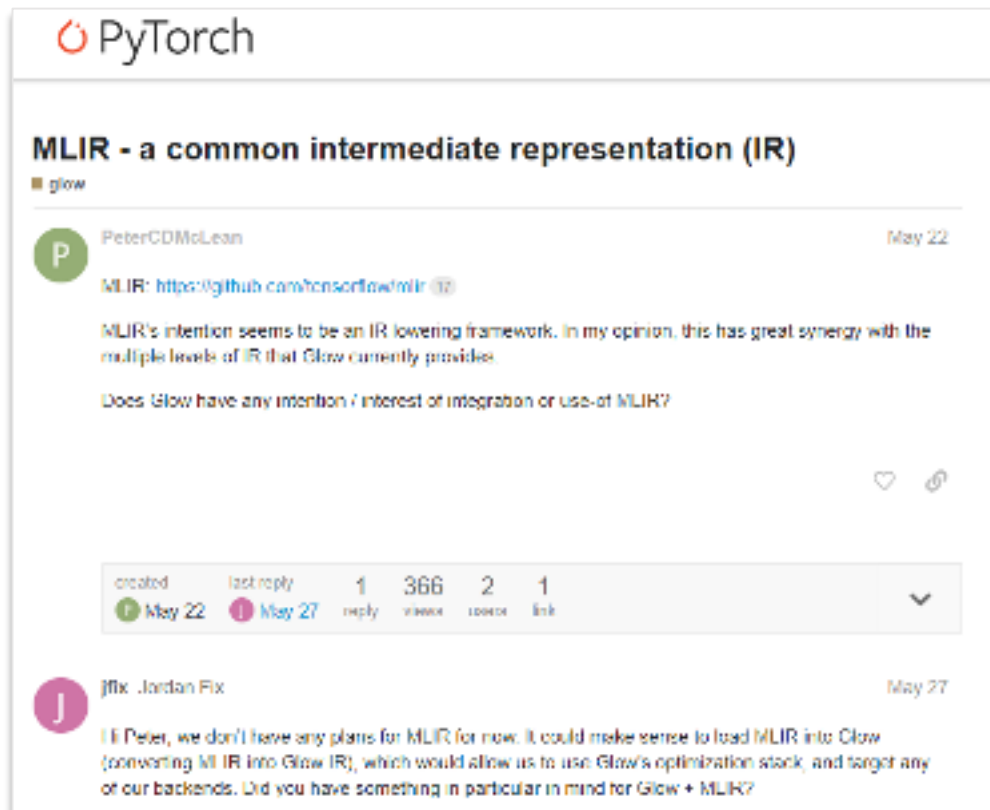
PyTorch

# Integration into existing frameworks is expensive

## Each framework has its own internal and external APIs

- No common code base
- Approaches such as MLIR, ONNX, DLPack, ... not widely adopted or very limited

# Integration into existing frameworks is expensive



The screenshot shows a GitHub issue on the PyTorch repository. The issue title is "MLIR - a common intermediate representation (IR)" and it is categorized under the "glow" label. The issue was created by PeterCDMcLean on May 22. The content of the issue asks if Glow has any intention or interest in integrating or using MLIR. A response from Jordan Fix (jfix) on May 27 states that there are no current plans for MLIR integration into Glow, but it would be possible to load MLIR into Glow and target backends.

**PyTorch**

## MLIR - a common intermediate representation (IR)

glow

**P** PeterCDMcLean May 22

MLIR: <https://github.com/tensorflow/mlir>

MLIR's intention seems to be an IR lowering framework. In my opinion, this has great synergy with the multiple levels of IR that Glow currently provides.

Does Glow have any intention / interest of integration or use-of MLIR?

created last reply 1 366 2 1  
May 22 May 27 reply views votes link

**J** jfix · Jordan Fix May 27

Hi Peter, we don't have any plans for MLIR for now. It could make sense to load MLIR into Glow (converting MLIR into Glow IR), which would allow us to use Glow's optimization stack, and target any of our backends. Did you have something in particular in mind for Glow + MLIR?

# Integration into existing frameworks is expensive

The image shows a collage of GitHub content related to MLIR support in PyTorch. At the top right is the PyTorch logo. Below it is a discussion titled "MLIR - a common intermediate representation (IR)" dated May 22. To the left is a GitHub issue titled "Any plans to support MLIR #1226" which is closed, opened by Arjuna197 24 days ago. The issue has one comment from dlibenzi, a collaborator, who explains that MLIR nodes have a `Lower()` virtual function and that integration will likely involve plugging MLIR behind the XLA builder.

PyTorch

MLIR - a common intermediate representation (IR)

May 22

g framework. In my opinion, this has great synergy with the video.

Integration or use-of MLIR?

2 1  
closed link

May 27

for now, it could make sense to load MLIR into Glow and allow us to use Glow's optimization stack, and target any particular in mind for Glow + MLIR?

Any plans to support MLIR #1226

Closed Arjuna197 opened this issue 24 days ago · 1 comment

Arjuna197 commented 24 days ago

Questions and Help

Do you have any plans to support pytorch->mlir?

dlibenzi commented 24 days ago Collaborator

Our IR nodes have a `Lower()` virtual function, which today lowers to have.

When MLIR will be stable enough, the first integration step for it would be likely to plug behind the XLA builder (the thing we use to lower to XLA), and generate MLIR behind the scenes.

Eventually, assuming MLIR will reach stability at some point, we will likely convert the XLA builder lowering to the proper MLIR counter-part.

# Integration into existing frameworks is expensive

The screenshot displays a GitHub pull request list for the repository 'amd-hip'. The search filter is set to 'amd-hip'. The list shows several pull requests, with the following details:

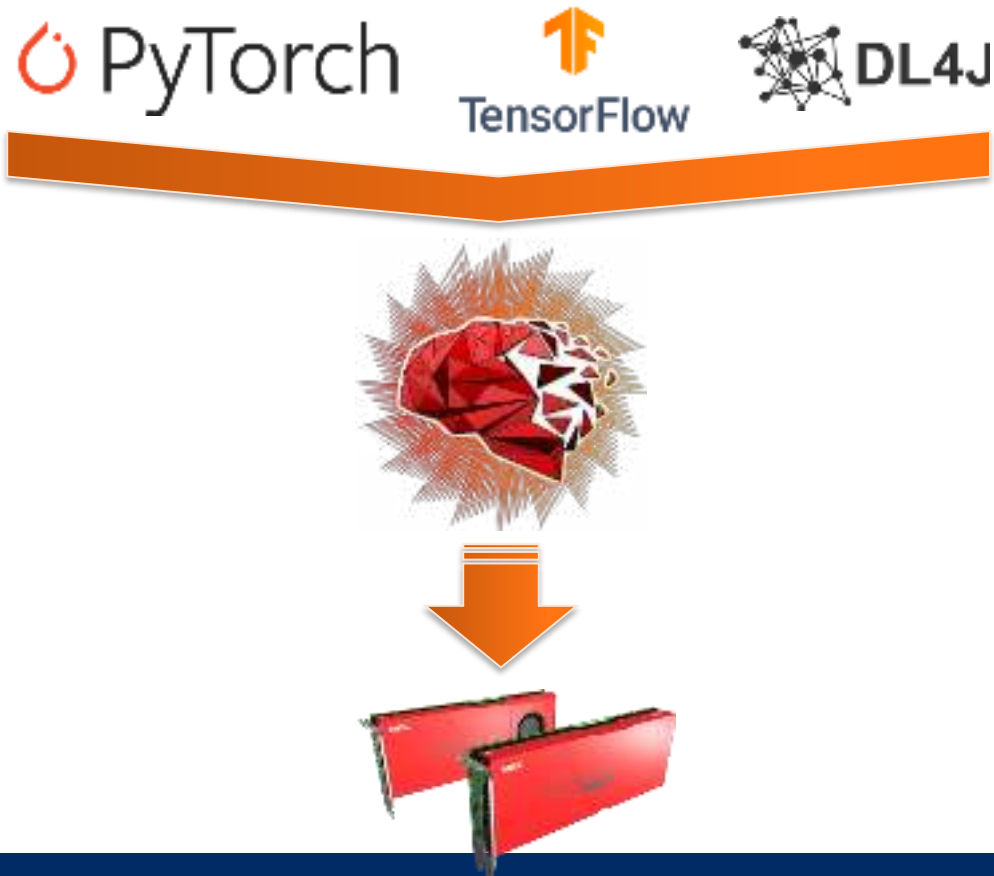
Issue Number	Author	Status	Labels	Projects	Milestones	Reviews	Assignee	Sort
#8306	petras	Merged (Jun 13, 2018)	Approved	open source		3		
#8257	lough12	Closed (Sep 21, 2018)				18		
#7566	petras	Merged (May 24, 2018)	Approved	open source		32		
#6836	daquexian	Merged (Jun 4, 2018)	Approved	caffe2, open source		22		
#6625	lough12	Merged (May 16, 2018)	Approved	open source		68		
#3544	wettiger	Closed (Nov 9, 2017)	Changes requested			4		
#2365	wettiger	Closed (Sep 18, 2017)				5		

Navigation: Previous | 1 | 2 | Next



## **SOL is a full stack AI acceleration middleware**

- Add-on to AI frameworks that does not require any code changes to the framework
- Optimizations range from mathematical/algorithmic down to actual implementations/code generation





## What data scientists see:

```
x = Conv(x, kernel=1x1, bias=True)
```

```
x = ReLU(x)
```

```
x = AvgPooling(x, kernel=13x13)
```

# SOL in a nutshell

## What data scientists see:

`x = Conv(x, kernel=1x1, bias=True)`

`x = ReLU(x)`

`x = AvgPooling(x, kernel=13x13)`



# SOL in a nutshell

## What data scientists see:

```
x = Conv(x, kernel=1x1, bias=True)
x = ReLU(x)
x = AvgPooling(x, kernel=13x13)
```

## What HPC people see:

`function(Conv):`

```
for(Batch, OutChannel, Y, X):
    for(InChannel, KernelY, KernelX):
        output[...] += input[...] * weight[...]
        output[...] += bias[...]
```

`function(ReLU):`

```
for(Batch, OutChannel, Y, X):
    output[...] = max(0, input[...])
```

`function(AvgPooling):`

```
for(Batch, OutChannel, Y, X):
    for(KernelY, KernelX):
        output[...] += input[...] / (13*13)
```



# SOL in a nutshell (continued)

## What we actually want:

```
function(FusedNetwork):  
    for(Batch, OutChannel):  
        float N[...]  
        for(Y, X):  
            for(InChannel, KernelY, KernelX):  
                N[...] += input[...] * weight[...]  
            N[...] += bias[...]  
            N[...] = max(0, X)  
        for(Y, X):  
            for(KernelY, KernelX):  
                output[...] += N[...] / (13*13)
```

# SOL in a nutshell (more continued)

## All layers merged into a single kernel function, using specialized hardware features

```
__global__ void F64486B08(...) {  
    const int 00idx = omp_get_thread_num();  
    const int 00 = 00idx / 256;  
    const int 01 = 00idx % 256;  
    float T64[169];  
    #pragma _NEC ivdep  
    for(int 02idx = 0; 02Idx < 169; 02Idx++) {  
        float T63 = 0.0f;  
        for(int I1 = 0; I1 < 512; I1++) {  
            T63 += T61[00 * 86528 + I1 * 169 + 02idx] * P63_weight[01 * 512 + I1];  
            T63 = (T63 + P63_bias[01]);  
            T64[02Idx] = sol_ncc_max(T63, 0.0f);  
        }  
        T66[01] = sol_ncc_reduce_add(T64);  
        T66[01] = (T66[01] / 169.0f);  
    }  
}
```

**Cores**

**Vector**

**inner loop**

**Reduction**

// #1 Convolution: 1x1 Pooling  
// #1 Convolution: Bias  
// #2 ReLU  
// #3 AvgPooling: 13x13 Pooling  
// #3 AvgPooling: Normalization

# SOL Usage (PyTorch)

```
import torch
from torchvision import models

py_model = models.__dict__["..."]()
input = torch.rand(1, 32, 224, 224)

output = py_model(input)
```

# SOL Usage (PyTorch)

```
import torch
from torchvision import models
import sol.pytorch as sol

py_model = models.__dict__["..."]()
input = torch.rand(1, 32, 224, 224)
sol_model = sol.optimize(py_model, input)
output = sol_model(input)
```

# How SOL integrates into the frameworks?

- SOL injects its optimized code as custom model into the framework

```
class SolLayer(torch.nn.Module):  
    def __init__(self):  
        self.ParamA = ...  
        self.ParamB = ...  
  
    def forward(self, X):  
        return sol.run(X, self.ParamA, self.ParamB)
```



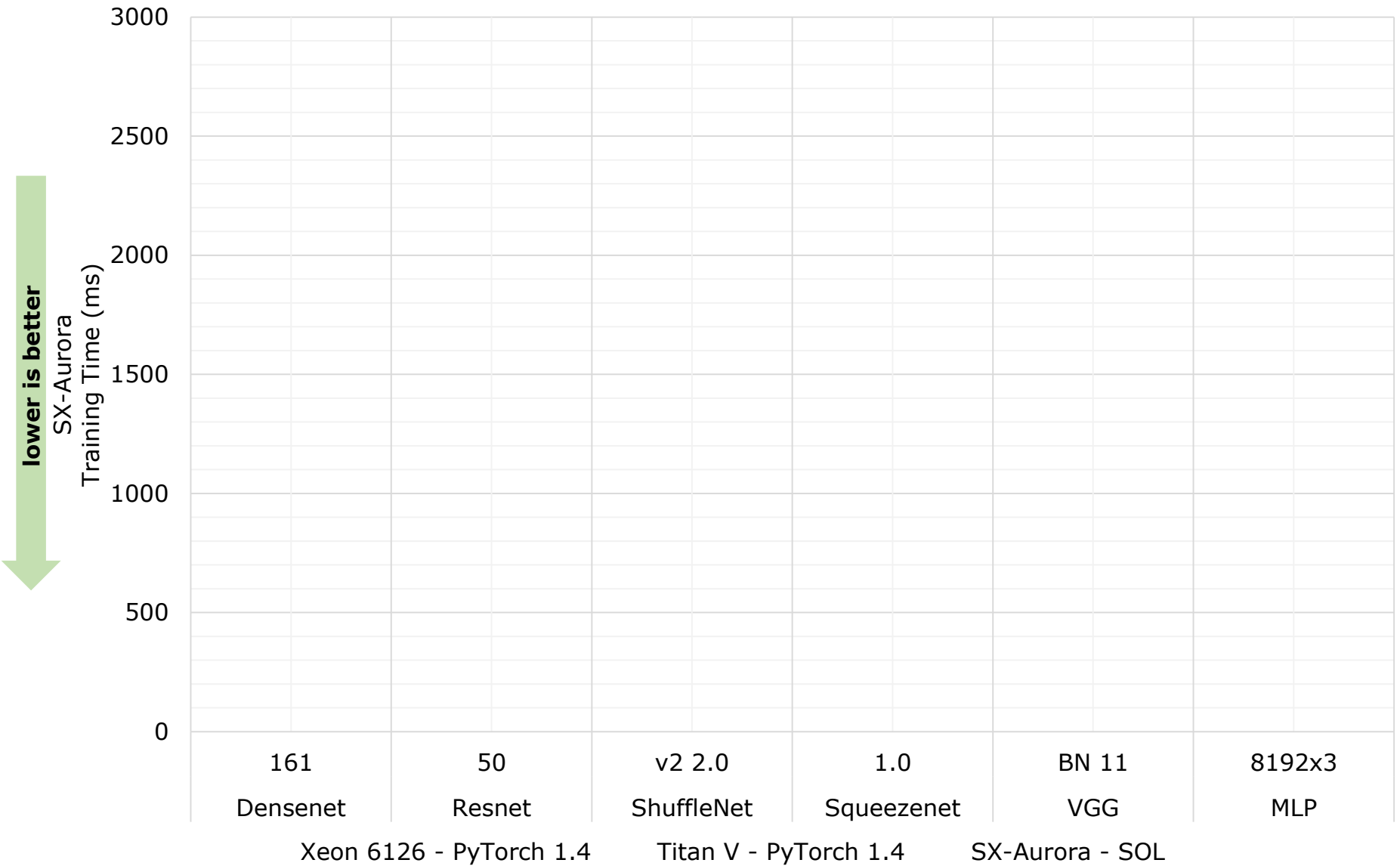
**framework handles  
model parameters!**



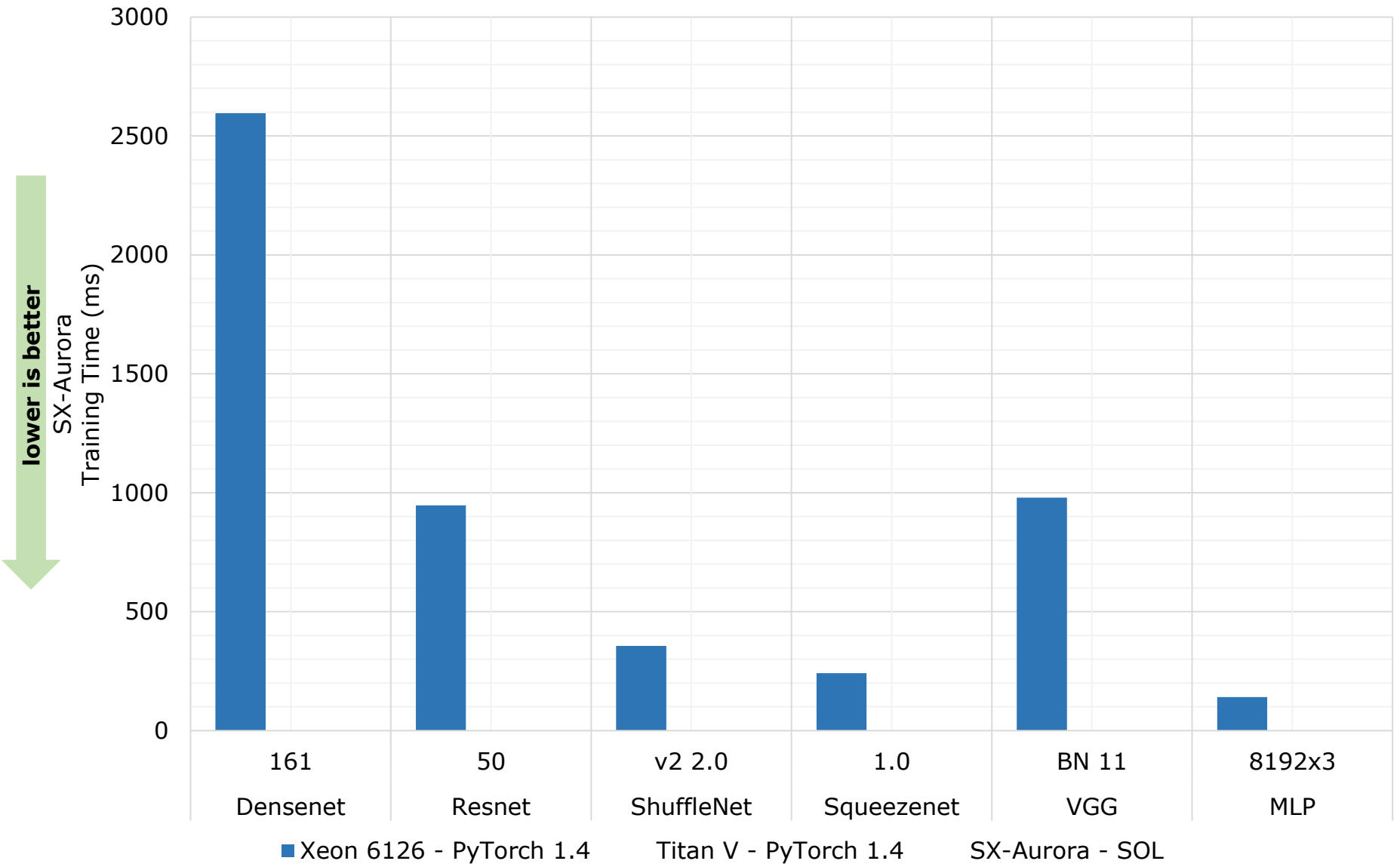
**SOL handles  
execution**



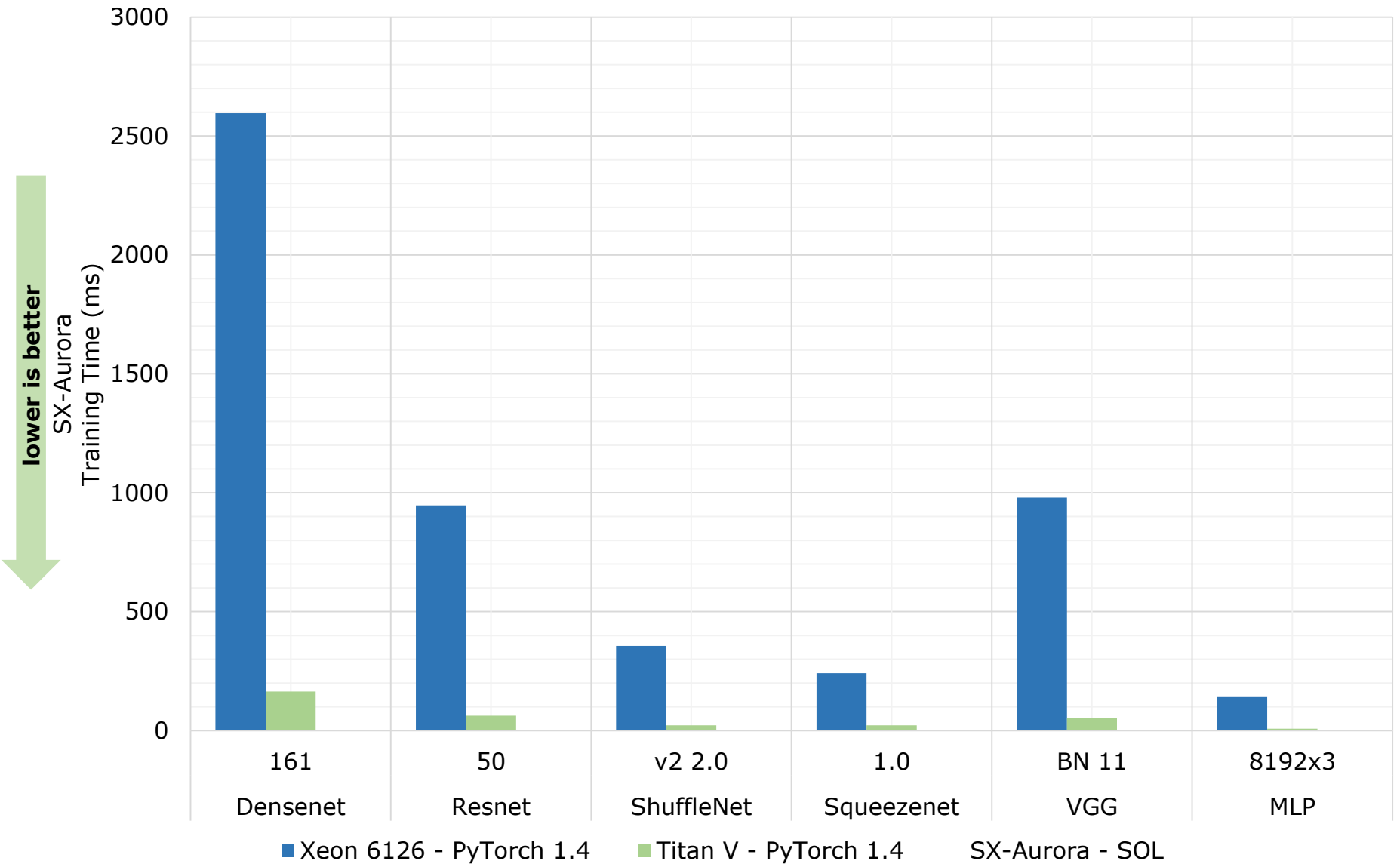
# Training Performance (CNN BS=16, MLP BS=64, FP32)



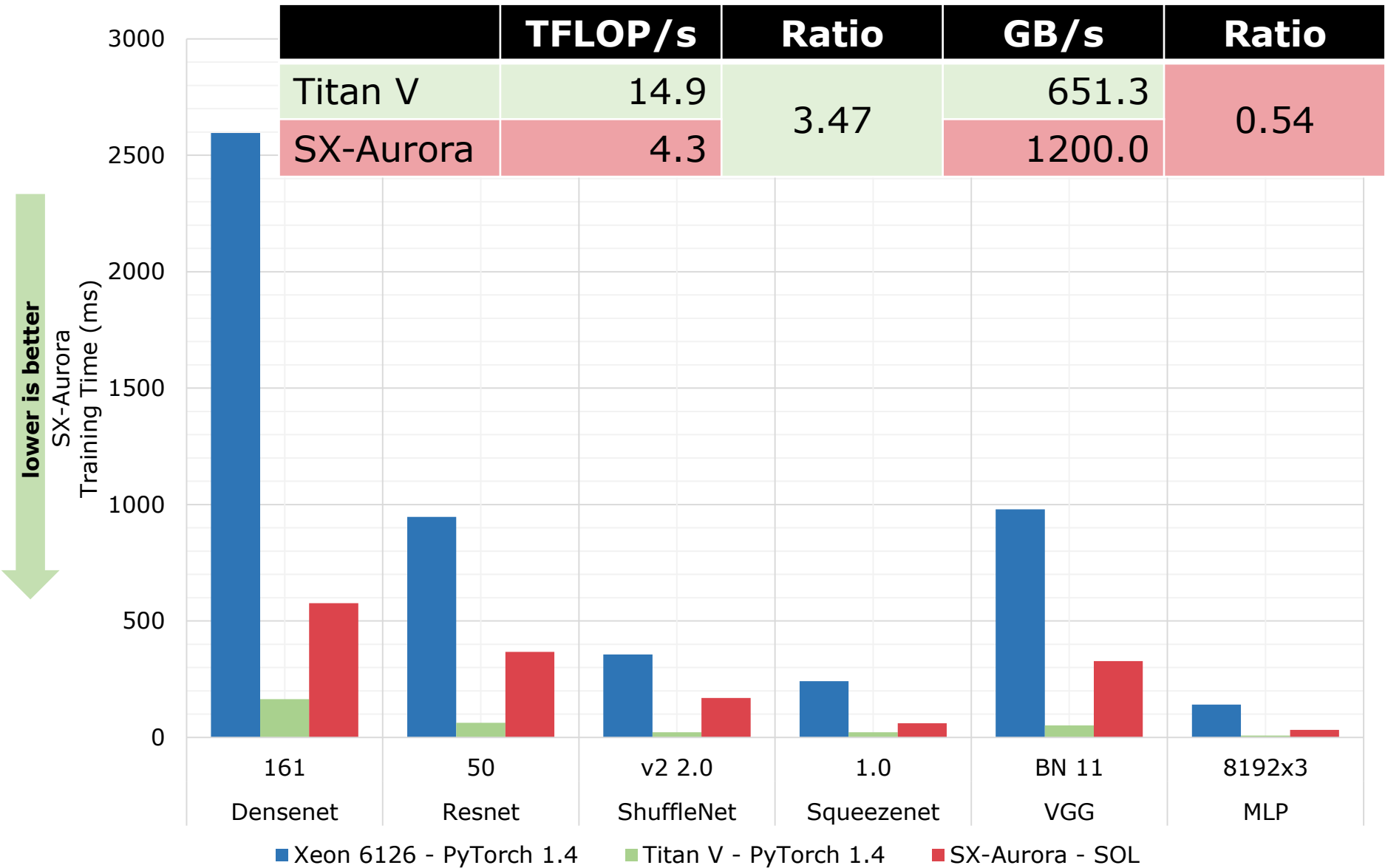
# Training Performance (CNN BS=16, MLP BS=64, FP32)



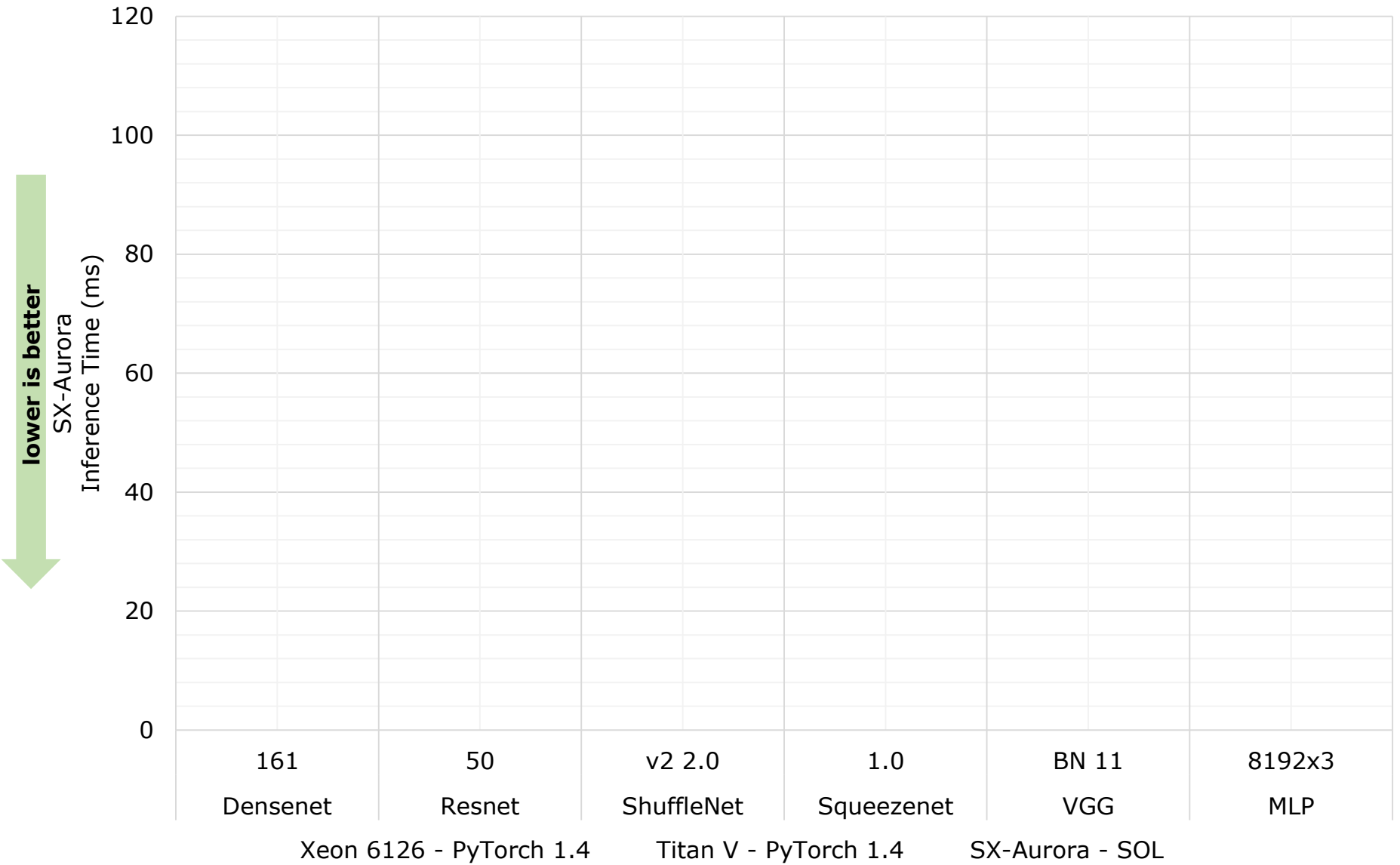
# Training Performance (CNN BS=16, MLP BS=64, FP32)



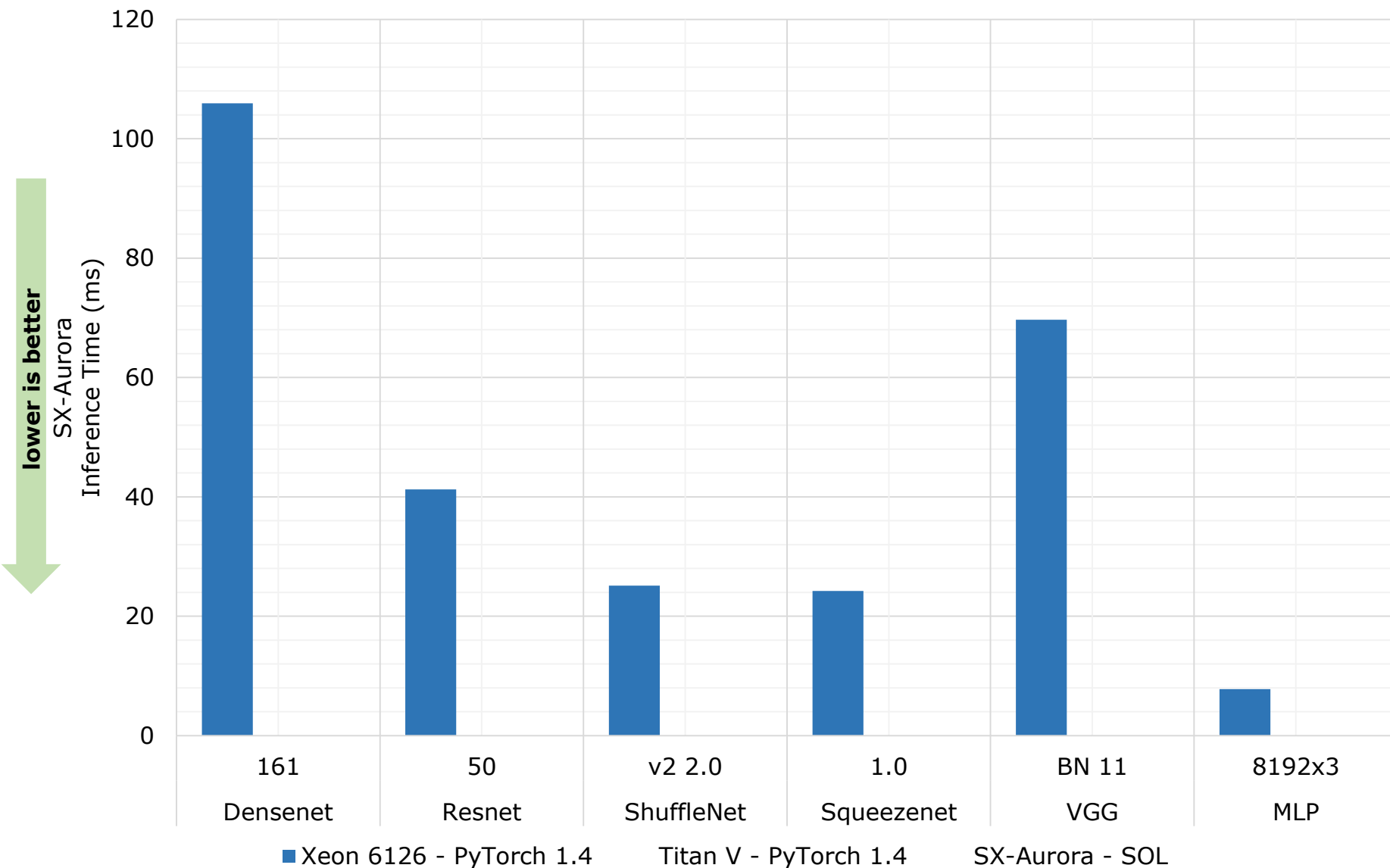
# Training Performance (CNN BS=16, MLP BS=64, FP32)



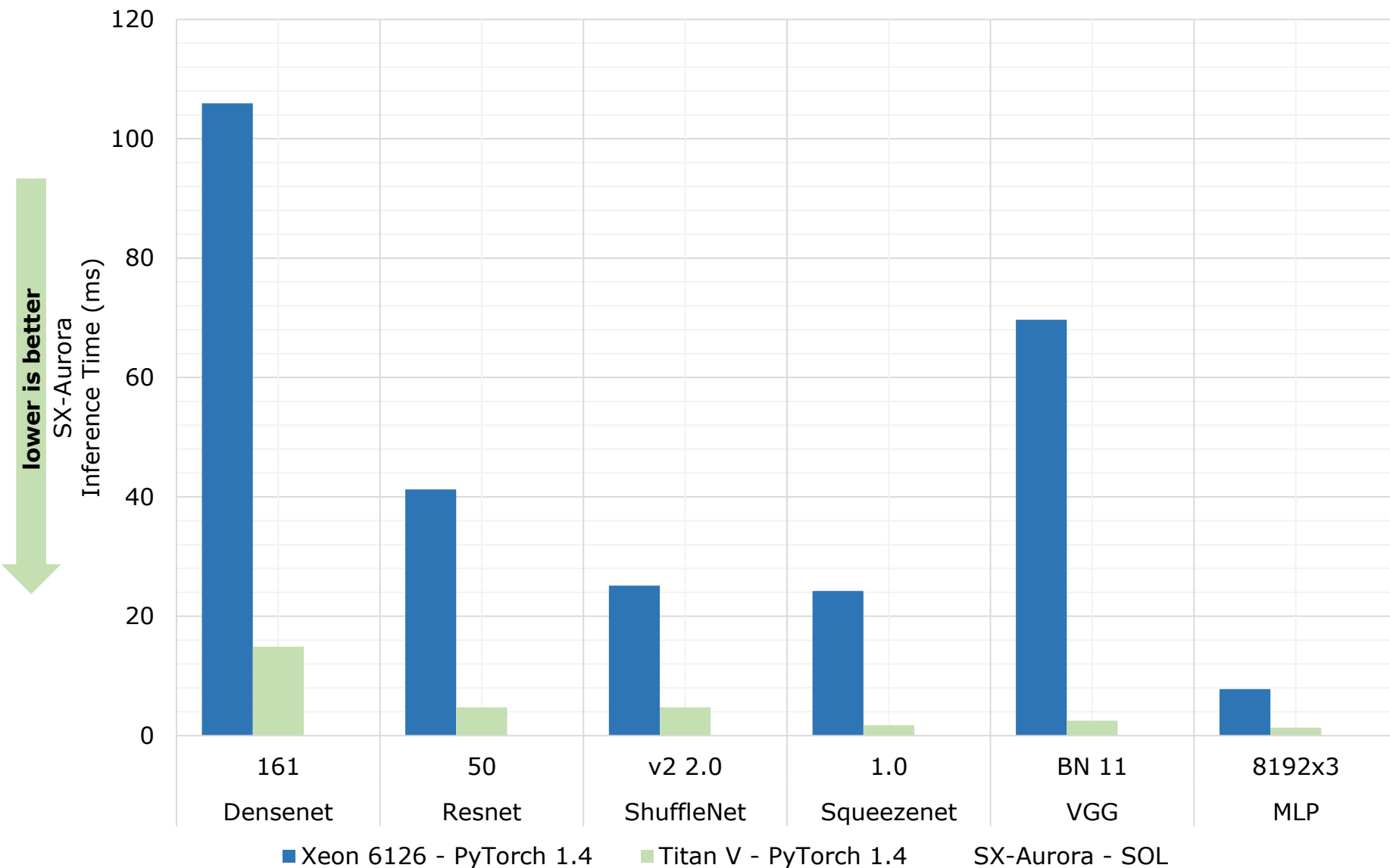
# Inference Performance (BS=1, FP32)



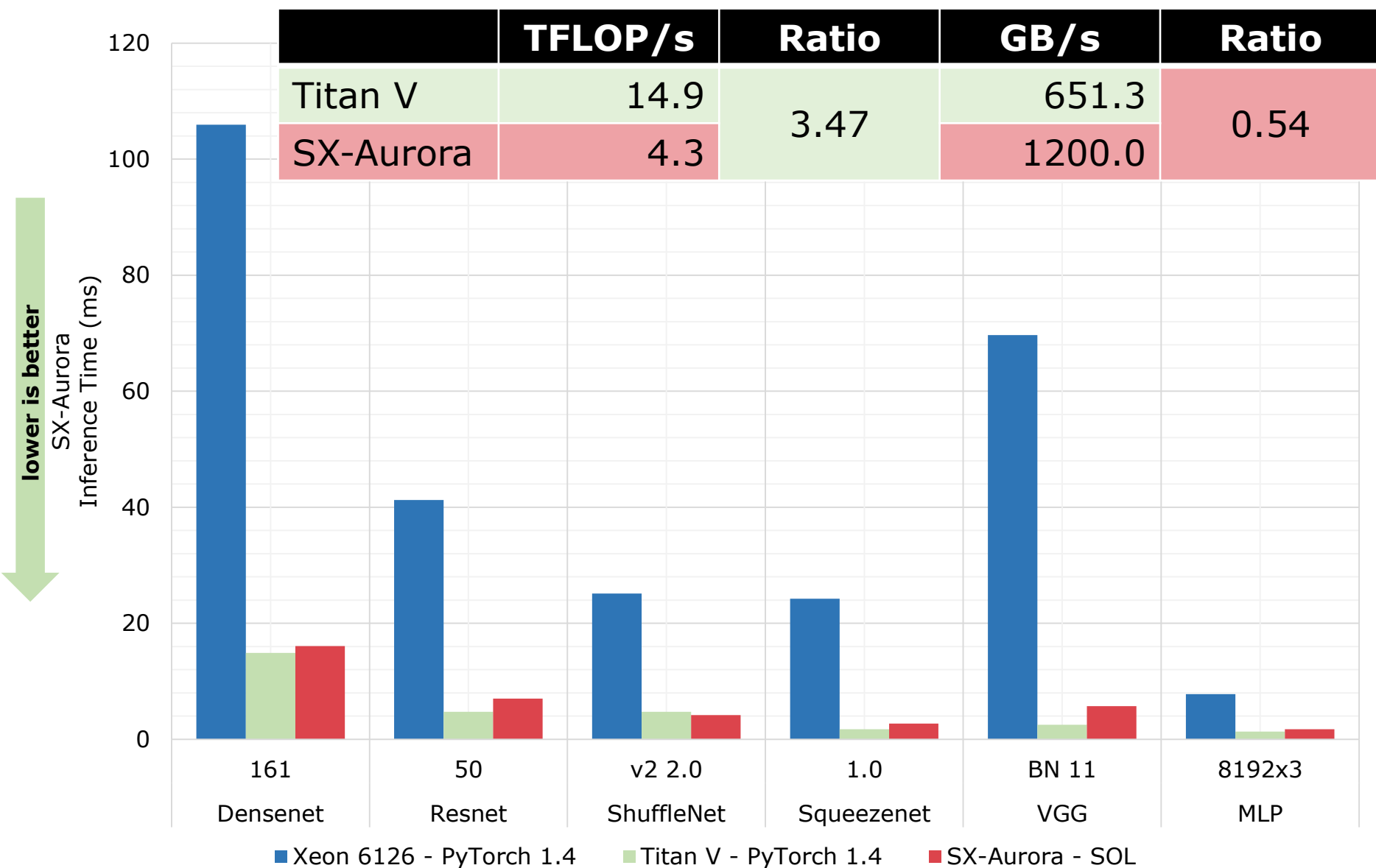
# Inference Performance (BS=1, FP32)



# Inference Performance (BS=1, FP32)



# Inference Performance (BS=1, FP32)





# How to use DNN in my own software?

## Again, dozen of available tools...

- TF-Lite
- LibTorch
- ONNXRuntime
- OpenVino (only Intel)
- NGraph
- TVM
- TensorRT (only NVIDIA)
- SOL
- ...

# How to use DNN in my own software?

```
sol.deploy(trained_model, [input],  
target=sol.deployment.shared_lib, device=sol.device.vc,  
lib_name="MyNetwork", func_name="predict", ...)
```

```
#ifndef __MyNetwork__  
#define __MyNetwork__
```

```
#ifdef __cplusplus  
extern "C" {  
#endif
```

```
void predict_init(const int deviceId);  
int predict_seed(const int seed);  
void predict      (void* ctx, const float* input, float** output);
```

```
#ifdef __cplusplus  
}  
#endif  
#endif
```

## **Status Quo:**

- PyTorch and ONNX
- CNN, MLP, Transformer, ...
- Training, Inference, Deployment
- ...

# SOL RoadMap: Tested Neural Networks

## Convolutional Neural Networks

- Alexnet
- SqueezeNet (1.0, 1.1)
- VGG + BN (11, 13, 16, 19)
- Resnet (18, 34, 50, 101, 152)
- Densenet (121, 161, 169, 201)
- Inception V3
- GoogleNet
- MobileNet (v1, v2)
- MNasNet (0.5, 0.75, 1.0, 1.3)
- ShuffleNet V2 (0.5, 1.0, 1.5, 2.0)
- ResNext (50, 101)
- WideResNet (50, 101)

## Multi Layer Perceptron (MLP)

## Linear/Logistic Regression

## Natural Language Processing

- BERT (PyTorchic + HuggingFace implemenations)
- *GPT-2 (in upcoming v0.3.0 release)*
- *LSTM+GRU (coming in Q4 2020)*

## **Status Quo:**

- PyTorch and ONNX
- CNN, MLP, Transformer, ...
- Training, Inference, Deployment
- ...

## **2020:**

- DL4J (October)
- TensorFlow v2 (December)
- Recurrent Neural Networks (LSTM, GRU)
- torch.nn.DataParallel support for PyTorch

## **2021:**

- Adjustable memory consumption during training (trading memory vs performance)
- User defined Custom Layers
- Algorithmic and internal code optimizations to improve performance
- NumPY support

Orchestrating a brighter world

**NEC**

# Frovedis

presented by Dr. Erich Focht, NEC-D

# Basics on SOL

# How to install

```
pip3 install sol-0.2.7.2-py3-none-any.whl
```

- enforces installation of dependencies

Coming in v0.3.0

```
pip3 install sol-0.3.0-py3-none-any.whl[torch, onnx]
```

- optional installation of dependencies (i.e. if you do not need support for all frameworks, etc.)



# SOL Vocabular

<b>Rest of the World</b>	<b>SOL</b>
Layer	Layer
Tensor	Tensor
Model/Neural Network	Model
Fused Layers	Cluster
Framework	Frontend
Device	Device
Compute Library/Compiler	Backend

## Importing SOL:

- import sol.pytorch as sol

```
[INFO ][ 0.00][core] Log (87) :  
[INFO ][ 0.00][core] Log (88) :  
[INFO ][ 0.00][core] Log (89) :  
[INFO ][ 0.00][core] Log (90) :  
[INFO ][ 0.00][core] Log (91) :  
[INFO ][ 0.00][core] Log (92) :  
[INFO ][ 0.00][core] Log (93) :  
[INFO ][ 0.00][core] Log (94) :  
[INFO ][ 0.00][core] Log (95) :  
[INFO ][ 0.00][core] Log (96) :  
[INFO ][ 0.00][core] Log (97) :  
[INFO ][ 0.00][core] Log (98) :  
[INFO ][ 0.00][core] Log (99) :  
[INFO ][ 0.00][core] Log (100) :  
  
#####  
SOL v0.3.0 Betelgeuse  
Copyright ©2020 NEC Laboratories Europe  
All rights reserved  
  
The use of this application requires explicit permit by NEC Laboratories  
Europe and is only allowed for demonstration purposes. Any redistribution  
in source or binary form, any modification or not explicitly authorized  
other use by NEC Laboratories Europe is strictly prohibited!  
#####
```

release name

SOL version

Disclaimer

Log Level

Time in seconds since start

SOL component

source location

# SOL Devices

sol.devices()

```
#####  
SOL Device Dump:
```

```
X86 CPUs
```

```
*[x86:0] Intel(R) Xeon(R) Gold 6126 CPU @ 2.60GHz, 12 cores
```

```
NEC SX-Aurora Vector Engine
```

```
*[ve:0] NEC SX-Aurora Tsubasa VE101, Firmware: 5399, 8 cores  
#####
```

```
#####  
SOL Device Dump:
```

```
X86 CPUs
```

```
*[x86:0] Intel(R) Xeon(R) Gold 6126 CPU @ 2.60GHz, 12 cores
```

```
NEC SX-Aurora Vector Engine
```

```
*[ve:0] NEC SX-Aurora Tsubasa VE101, Firmware: 5399, 8 cores, 24B/48.00G  
#####
```

star indicates default device

activated device

currently used memory

# SOL Versions

sol.versions()

#####  
**SOL Version Dump:**

```
AVEO          0.9.12
DNNL          1.6.0
GraphVIZ      2.30.1
ISPC          1.14.1
Linux        CentOS Linux 7 (Core), 3.10.0-1127.13.1.el7.x86_64
MKL           2020.0.1
NEC NAR       2.26.20160125
NEC NC++      3.0.28
NEC NLD       2.26.20160125
NNPACK        bundled
OneTBB        2020_U3
PyTorch       1.6.0
Python        3.6.9
SOL           0.3.0, Betelgeuse
SQLite        3.32.3
VEASL         2.1.0
VEBLAS        2.1.0
VEDA          linked: 0.9.3, loaded: 0.9.3
VEDNN         bundled
VEOS          2.5.0
X86 GCC AR    2.30
X86 GCC G++   8.3.1
X86 GCC GCC   8.3.1
X86 GCC LD    2.30
```

## Print Seeds:

- `sol.seeds()`

```
#####  
SOL Seed Dump: (0x5F71BC4C / 1601289292)  
X86 CPUs (0x5F71BC4C / 1601289292)  
[x86:0] 0x00000000 / 0  
NEC SX-Aurora Vector Engine (0x5F71BC4C / 1601289292)  
[ve:0] 0x5F71BC4C / 1601289292  
#####
```

## 3 Types of Seeds:

- Global (all devices)
- DeviceType (all devices of same type)
- Device (a specific device)

## Get seed:

- `sol.seed(deviceType=None, deviceIdx=None)`
- `sol.seed(deviceType=sol.device.ve, deviceIdx=None)`
- `sol.seed(deviceType=sol.device.ve, deviceIdx=0)`

## Set seed:

- `sol.set_seed(seed, deviceType=None, deviceIdx=None)`
- `sol.set_seed(seed, deviceType=sol.device.ve, deviceIdx=None)`
- `sol.set_seed(seed, deviceType=sol.device.ve, deviceIdx=0)`

# Debugging

sol.config[“compiler::name”] = “Prefix Used for Debugging Output”

C/C++ device code generated in .sol/ve/source

- Might not be obvious to read

sol.config[“compiler::debug”] = True

- Compiles with debug symbols
- Prints execution times of fused layers
- Outputs visualized NN in .sol/debug/ subfolder
- Requires: GraphViz (Dot)

# Debugging

sol.config["compiler::debug"] = True

- Prints execution times of fused layers

```
ve_0EAE7ED7_vednn_FI_385 76.912 µs
ve_0EAE7ED7_ncc_FI_38B 0.500 µs
ve_0EAE7ED7_vednn_FI_38E 0.029 µs
ve_0EAE7ED7_ncc_FI_394 0.020 µs
ve_0EAE7ED7_vednn_FI_3A0 0.039 µs
ve_0EAE7ED7_ncc_FI_481 0.036 µs
ve_0EAE7ED7_vednn_FI_397 0.025 µs
ve_0EAE7ED7_ncc_FI_47E 0.037 µs
ve_0EAE7ED7_vednn_FI_3A9 0.033 µs
ve_0EAE7ED7_ncc_FI_3AF 0.019 µs
ve_0EAE7ED7_vednn_FI_3BB 0.038 µs
ve_0EAE7ED7_ncc_FI_487 0.036 µs
ve_0EAE7ED7_vednn_FI_3B2 0.023 µs
ve_0EAE7ED7_ncc_FI_484 0.037 µs
ve_0EAE7ED7_ncc_FI_3C4 0.038 µs
ve_0EAE7ED7_vednn_FI_3C7 0.022 µs
ve_0EAE7ED7_ncc_FI_3CD 0.017 µs
ve_0EAE7ED7_vednn_FI_3D9 0.037 µs
ve_0EAE7ED7_ncc_FI_48D 0.025 µs
ve_0EAE7ED7_vednn_FI_3D0 0.022 µs
ve_0EAE7ED7_ncc_FI_48A 0.025 µs
ve_0EAE7ED7_vednn_FI_3E2 0.026 µs
ve_0EAE7ED7_ncc_FI_3E8 0.017 µs
ve_0EAE7ED7_vednn_FI_3F4 0.035 µs
ve_0EAE7ED7_ncc_FI_493 0.025 µs
ve_0EAE7ED7_vednn_FI_3EB 0.023 µs
ve_0EAE7ED7_ncc_FI_490 0.025 µs
ve_0EAE7ED7_ncc_FI_3FD 0.021 µs
ve_0EAE7ED7_vednn_FI_400 0.024 µs
ve_0EAE7ED7_ncc_FI_406 0.015 µs
ve_0EAE7ED7_vednn_FI_412 0.032 µs
ve_0EAE7ED7_ncc_FI_499 0.018 µs
ve_0EAE7ED7_vednn_FI_409 0.022 µs
ve_0EAE7ED7_ncc_FI_496 0.018 µs
ve_0EAE7ED7_vednn_FI_41B 0.024 µs
ve_0EAE7ED7_ncc_FI_421 0.015 µs
ve_0EAE7ED7_vednn_FI_42D 0.032 µs
```

## Index of /v0.2.7.2/.sol/ve/src/

../			
<a href="#">ve_0EAE7ED7.h</a>	28-Sep-2020 12:57	2763	
<a href="#">ve_0EAE7ED7_BT.cpp</a>	28-Sep-2020 12:57	20K	
<a href="#">ve_0EAE7ED7_BT.o</a>	28-Sep-2020 12:57	81K	
<a href="#">ve_0EAE7ED7_FI.cpp</a>	28-Sep-2020 12:57	14K	
<a href="#">ve_0EAE7ED7_FI.o</a>	28-Sep-2020 12:57	75K	
<a href="#">ve_0EAE7ED7_FT.cpp</a>	28-Sep-2020 12:57	14K	
<a href="#">ve_0EAE7ED7_FT.o</a>	28-Sep-2020 12:57	76K	
<a href="#">ve_0EAE7ED7_ve.cpp</a>	28-Sep-2020 12:57	1054	
<a href="#">ve_0EAE7ED7_ve.o</a>	28-Sep-2020 12:57	26K	
<a href="#">ve_0EAE7ED7_ve_0EAE7ED7_ncc_BT_38D.cpp</a>	28-Sep-2020 12:57	2511	
<a href="#">ve_0EAE7ED7_ve_0EAE7ED7_ncc_BT_38D.o</a>	28-Sep-2020 12:57	52K	
<a href="#">ve_0EAE7ED7_ve_0EAE7ED7_ncc_BT_396.cpp</a>	28-Sep-2020 12:57	1889	
<a href="#">ve_0EAE7ED7_ve_0EAE7ED7_ncc_BT_396.o</a>	28-Sep-2020 12:57	44K	
<a href="#">ve_0EAE7ED7_ve_0EAE7ED7_ncc_BT_3B1.cpp</a>	28-Sep-2020 12:57	1912	
<a href="#">ve_0EAE7ED7_ve_0EAE7ED7_ncc_BT_3B1.o</a>	28-Sep-2020 12:57	44K	
<a href="#">ve_0EAE7ED7_ve_0EAE7ED7_ncc_BT_3C6.cpp</a>	28-Sep-2020 12:57	1454	
<a href="#">ve_0EAE7ED7_ve_0EAE7ED7_ncc_BT_3C6.o</a>	28-Sep-2020 12:57	41K	
<a href="#">ve_0EAE7ED7_ve_0EAE7ED7_ncc_BT_3CF.cpp</a>	28-Sep-2020 12:57	1910	
<a href="#">ve_0EAE7ED7_ve_0EAE7ED7_ncc_BT_3CF.o</a>	28-Sep-2020 12:57	44K	
<a href="#">ve_0EAE7ED7_ve_0EAE7ED7_ncc_BT_3EA.cpp</a>	28-Sep-2020 12:57	1910	
<a href="#">ve_0EAE7ED7_ve_0EAE7ED7_ncc_BT_3EA.o</a>	28-Sep-2020 12:57	44K	
<a href="#">ve_0EAE7ED7_ve_0EAE7ED7_ncc_BT_3FF.cpp</a>	28-Sep-2020 12:57	1453	
<a href="#">ve_0EAE7ED7_ve_0EAE7ED7_ncc_BT_3FF.o</a>	28-Sep-2020 12:57	41K	
<a href="#">ve_0EAE7ED7_ve_0EAE7ED7_ncc_BT_408.cpp</a>	28-Sep-2020 12:57	1880	

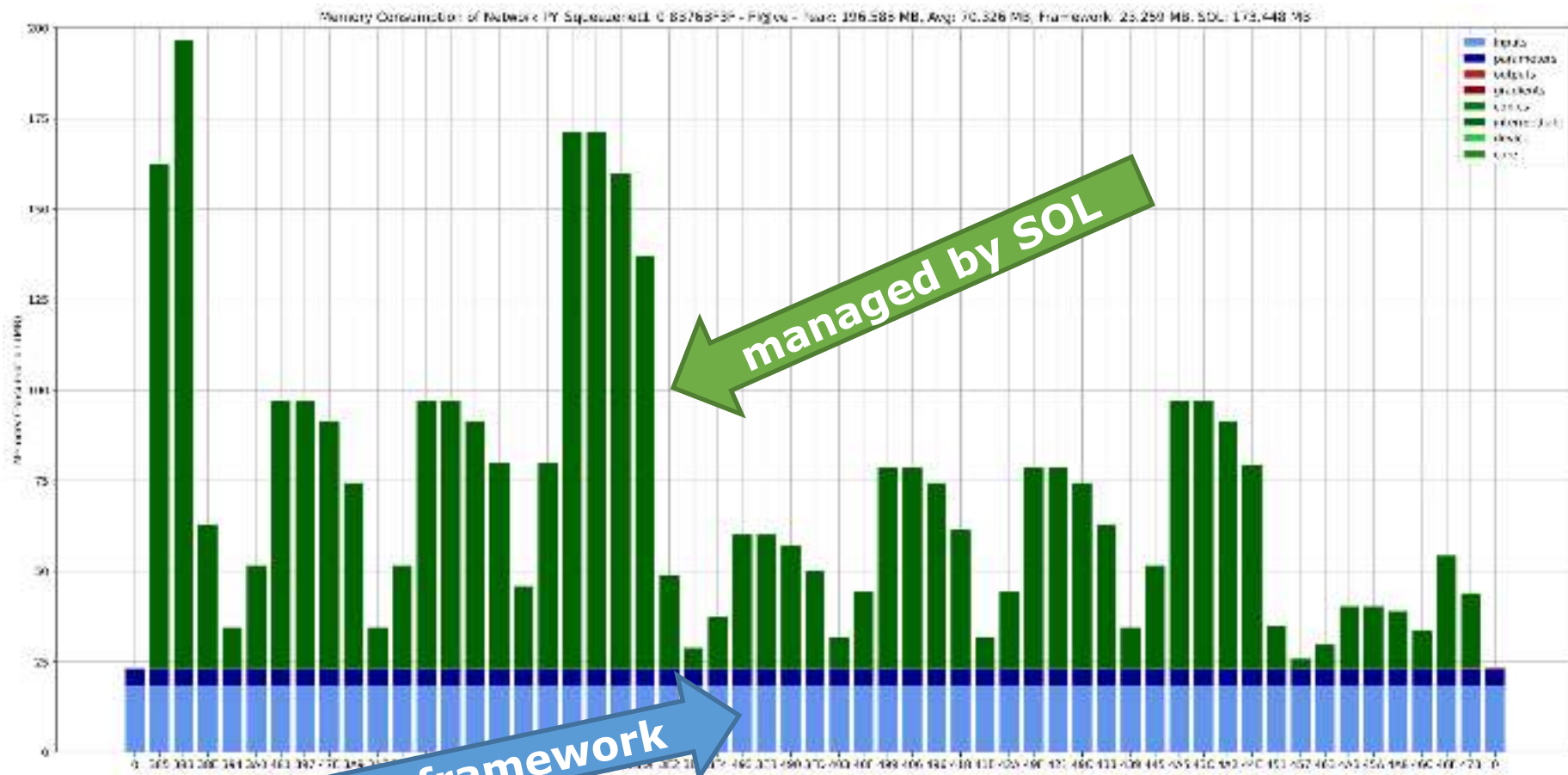




# Debugging

```
sol.config["compiler::debug_memory_consumption"]=True
```

- Outputs memory consumption plots
- Requires: matplotlib



# Debugging

`sol.config["compiler::name"] = "Prefix Used for Debugging Output"`

C/C++ device code generated in `.sol/ve/source`

- Might not be obvious to read

`sol.config["compiler::debug"] = True`

- Compiles with debug symbols
- Prints execution times of fused layers
- Outputs visualized NN in `.sol/debug/` subfolder
- Outputs memory consumption plots
- Requires: matplotlib, GraphViz (Dot)

Activate tracing:

- `sol.config["log::level"] = sol.log.[error, info, warn, debug, trace]`
- `SOL_LOG=TRACE python3 mySolScript.py`

# SOL's VE integration into PyTorch

# SOL's VE integration into PyTorch

**PyTorch does not come with support for storing data on VE devices.**

**SOL adds this support into PyTorch automatically when loaded.**

**We misuse the HIP-device for the VE's as we can't add new device types without recompiling PyTorch:**

- see <https://arxiv.org/abs/2003.10688> for details

## SOL: Effortless Device Support for AI Frameworks without Source Code Changes

Nicolas Weber and Felipe Huici  
*NEC Laboratories Europe*

*Abstract*—Modern high performance computing clusters heavily rely on accelerators to overcome the limited compute power of CPUs. These supercomputers run various applications from different domains such as simulations, numerical applications or artificial intelligence (AI). As a result, vendors need to be able to

State of the Art	Proposed with SOL
API (Python, C/C++, ...)	API (Python, C/C++, ...)
Framework Core	Framework Core
Device Backends	SOL

# SOL's VE integration into PyTorch

**Identical to how CUDA is used in PyTorch, just with 'hip'**

**Copy data to VE:** `tensor_ve = tensor_cpu.to('hip:0')`

**Copy data to CPU:** `tensor_cpu = tensor_ve.cpu()` or  
`.to('cpu')`

**Copy model to VE:** `model.to('hip:0')`

**Unfortunately** `tensor.hip()` **does not work :(**

**Synchronize VE execution:**

- `torch.hip.synchronize()`

**Selection of VE's in Server**

- `export VEDA_VISIBLE_DEVICES=0,1,2`
- `export VEDA_VISIBLE_DEVICES=$VE_NODE_NUMBER`

# Known Issues

## `torch.concat()` on CPU can produce wrong results when SOL4VE is loaded

- Submitted bugfix to PyTorch, was released in PyTorch v1.6.0. SOL v0.3.0 will support PyTorch v1.6.0.

## Only minimal number of functions implemented

- $A + B$ ,  $A - B$ , `print(A)`, ...
- Otherwise you will get a message like: "Function X not implemented for HipTensorId".
- Workaround:
  - `A.cpu().notImplemented().to('hip:0')`
- CAN ONLY OCCUR OUTSIDE OF YOUR NEURAL NETWORK!!!

## `print(tensor)` always shows scientific notation.

Orchestrating a brighter world

**NEC**

We finally want to use it!!!

# SOL Execution Modes

PyTorch supports four execution modes, SOL only two:

	<b>model.eval()</b>	<b>model.training()</b>
<b>torch.no_grad()</b>	SOL Inference	N/A
<del>torch.no_grad()</del>	N/A	SOL Training



# Optimizing a model

```
sol_model = sol.optimize(model, input0, input1,  
input2, ..., batch_size=32)
```

```
model = any torch.nn.Module
```

```
inputX
```

- torch.Tensor
- any primitive datatype (int, float, ...)
- sol.input([0, 3, 224, 224], requires\_grad=False, dtype=torch.float)
  - Size of 0 is a wildcard (only in first dimension!)

```
batch_size → needs to be set if wildcard is used, otherwise  
ignored. Is used by SOL in its heuristics.
```

# Model Preparation

```
import torch
import sol.pytorch as sol

class Model(torch.nn.Module):
    def forward(self, A, B):
        return A + B

py_model = Model()
sol.config[...] = ... # always set BEFORE sol.optimize
sol_model = sol.optimize(py_model, sol.input([0, 50]),
sol.input([0, 50]), batch_size=32)
sol_model.load_state_dict(py_model.state_dict())
sol_model.to('hip:0')
```

# Inference

```
# generate random input
A_cpu = torch.rand(5, 50)
B_cpu = torch.rand(5, 50)

# copy to VE
A_ve, B_ve = A_cpu.to('hip:0'), B_cpu.to('hip:0')

# activate inference mode
sol_model.eval()
with torch.no_grad():
    # run model
    C_ve = sol_model(A_ve, B_ve)
    # print result
    print(C_ve)
```

# Training

```
sol_model.training()
for epoch in range(epochs):
    for batch in train_data_loader:
        # get batch and copy to VE
        A_cpu, B_cpu = *batch
        A_ve, B_ve = A_cpu.to('hip:0'), B_cpu.to('hip:0')

        # run forward pass
        C_ve = sol_model(A_ve, B_ve)

        # compute loss on CPU
        C_cpu = C_ve.cpu()
        loss = loss_function(C_cpu)

        # run backward pass
        loss.backward()

        # Optional: wait for VE to complete this iteration
        torch.hip.synchronize()
```

## “**SQLITE Error UNIQUE CONSTRAINT ...**”

- SOL cache got corrupted. Either:
  - run: `rm -r .sol`
  - or call `sol.cache.clear()` before `sol.optimize(...)`

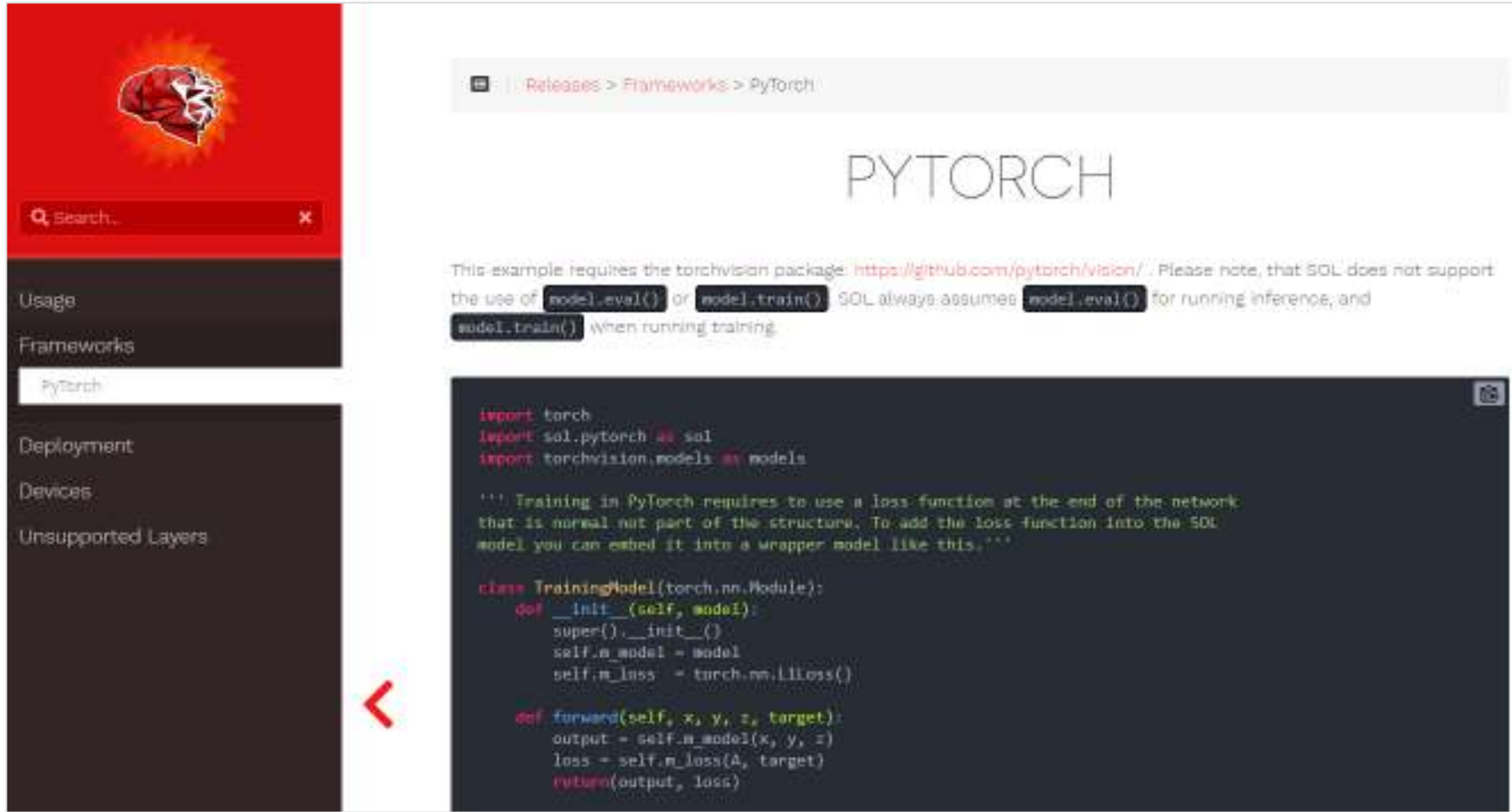
## **SOL does not complain when the model and the input data are not located on the same device:**

- fixed in v0.3.0

## **sol.deploy(...) not fully working in v0.2.7.2. Would need some manual fixing in generated code.**

- fixed in v0.3.0

# More information in the SOL docs



Releases > Frameworks > PyTorch

## PYTORCH

This example requires the torchvision package: <https://github.com/pytorch/vision/>. Please note, that SOL does not support the use of `model.eval()` or `model.train()`. SOL always assumes `model.eval()` for running inference, and `model.train()` when running training.

```
import torch
import sol.pytorch as sol
import torchvision.models as models

''' Training in PyTorch requires to use a loss function at the end of the network
that is normal not part of the structure. To add the loss function into the SOL
model you can embed it into a wrapper model like this. '''

class TrainingModel(torch.nn.Module):
    def __init__(self, model):
        super().__init__()
        self.m_model = model
        self.m_loss = torch.nn.L1Loss()

    def forward(self, x, y, z, target):
        output = self.m_model(x, y, z)
        loss = self.m_loss(A, target)
        return(output, loss)
```

# How to get started on ICM

```
# login to server  
ssh hpc.icm.edu.pl  
...
```

```
# install and activate virtualenv  
pip3 install --user virtualenv  
virtualenv sol  
source sol/bin/activate
```

```
# install sol  
pip3 install /apps/nec/sol/sol-0.2.7.2-py3-none-any.whl  
pip3 install torchvision==0.6.1
```

```
# test sol  
mkdir tmp  
cd tmp  
VEDA_VISIBLE_DEVICES=0 python3 /apps/nec/sol/test.py
```

# How to get started on ICM

```
[scl] kdmsrk20@pbaran ~/sol/rwp $ VEDA_VISIBLE_DEVICES=0 python3 test.py
[INFO] [[ 0.00][core] Log (90):
[INFO] [[ 0.00][core] Log (91):
[INFO] [[ 0.00][core] Log (92):
[INFO] [[ 0.00][core] Log (93):
[INFO] [[ 0.00][core] Log (94):
[INFO] [[ 0.00][core] Log (95):
[INFO] [[ 0.00][core] Log (96):
[INFO] [[ 0.00][core] Log (97):
[INFO] [[ 0.00][core] Log (98):
[INFO] [[ 0.00][core] Log (99):
[INFO] [[ 0.00][core] Log (100):
[INFO] [[ 0.00][core] Log (101):
[INFO] [[ 0.00][core] Log (102):
[INFO] [[ 0.00][core] Log (103):
[WARN] [[ 0.52][git-dot] Dot (13):
[INFO] [[ 0.55][core] NetworkBuilder (281):
[VE] ERROR: getsym_handler() dLError: ./sol/ve/69FF68A9.vso: undefined symbol: ve_69FF68A9_FT
[VE] ERROR: getsym_handler() dLError: ./sol/ve/69FF68A9.vso: undefined symbol: ve_69FF68A9_BT
CPU tensor([[1.6391, 0.8292, 1.0491, 1.4602, 0.1355],
           [1.2085, 0.8560, 1.2459, 1.1432, 0.4776],
           [1.1108, 0.8079, 1.0278, 1.3681, 0.7307],
           [1.3812, 1.3908, 0.9743, 0.5613, 0.7339],
           [0.9672, 0.2896, 0.6323, 1.2249, 0.8866]])
VE tensor([[1.6391e+00, 8.2916e-01, 1.0431e+00, 1.4602e+00, 1.3546e-01],
           [1.2085e+00, 8.5602e-01, 1.2459e+00, 1.1432e+00, 4.7764e-01],
           [1.1108e+00, 8.0732e-01, 1.0278e+00, 1.3681e+00, 7.3071e-01],
           [1.3812e+00, 1.3908e+00, 9.7426e-01, 5.6132e-01, 7.3390e-01],
           [9.6719e-01, 2.8963e-01, 6.3231e-01, 1.2249e+00, 8.8650e-01]],
          device='hip:0')
```



**Dr. Nicolas Weber**

Intelligent Software Systems Group

*Senior Software Engineer*

NEC Laboratories Europe

**nicolas.weber@neclab.eu**



**www.sol-project.org**

