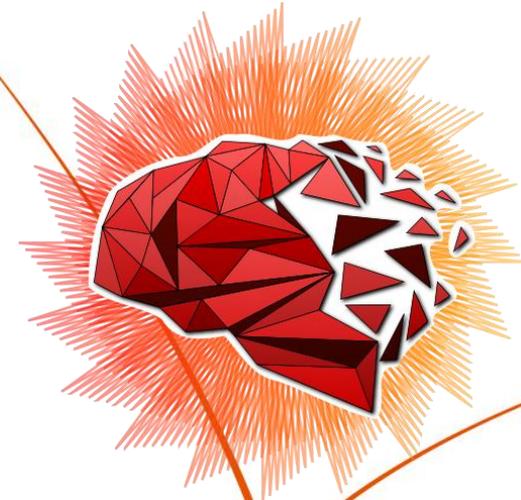


Orchestrating a brighter world

NEC



SOL4VE: Bringing Deep Neural Networks to the NEC SX-Aurora TSUBASA

Dr. Nicolas Weber (NEC Labs Europe)

■ **“[BuzzWord] is like teenage sex. Everybody wants it, but no one knows how to do it!”**

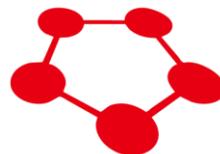
Motivation

“[BuzzWord] is like teenage sex. Everybody wants it, but no one knows how to do it!”

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



MLIR - a common intermediate representation (IR)

■ glow

 **PeterCDMcLean** May 22

MLIR: <https://github.com/tensorflow/mlir> ¹⁷

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	users	link	

 **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

PyTorch

MLIR - a common intermediate representation (IR)

May 22

g framework. In my opinion, this has great synergy with the
vides.
Integration or use-of MLIR?

2 users 1 link

May 27

for now. It could make sense to load MLIR into Glow
d allow us to use Glow's optimization stack, and target any
n 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?

dlbenzi 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 shows a GitHub pull request list for the repository 'amd hip'. The search filters are set to 'amd hip'. There are 171 labels and 1 milestone. A 'New pull request' button is visible in the top right. The list contains several pull requests, mostly marked as closed or merged. The pull requests are:

- #8306 [Caffe2] Enable AMD/MIOpen ops for Caffe2 (open source) - Merged on Jun 13, 2018. Approved. 3 comments.
- #8257 PyTorch/Caffe2 Build Path Changes (ROCm path) + MIOpen Integration (open source) - Closed on Sep 21, 2018. 18 comments.
- #7566 [Caffe2] Enabling AMD GPU Backend for Caffe2 (open source) - Merged on May 24, 2018. Approved. 32 comments.
- #6896 [Caffe2] Support non peer access in muji and fix bug when reduced_affix is empty (caffe2 open source) - Merged on Jun 4, 2018. Approved. 22 comments.
- #6625 PyTorch AMD Build Scripts (open source) - Merged on May 16, 2018. Approved. 68 comments.
- #3544 Initial commit for PyTorch to run on AMD hardware using the ROCM software stack (open source) - Closed on Nov 9, 2017. Changes requested. 4 comments.
- #2365 [WIP] Initial HIP-ification of PyTorch code for AMD hardware (open source) - Closed on Sep 18, 2017. 5 comments.

Navigation controls at the bottom show 'Previous', '1', '2' (selected), and 'Next'.

Integration into existing frameworks is expensive

The image shows a screenshot of a GitHub issue and pull request interface. The main focus is a GitHub issue titled "at::native::copy_impl causes a SegFault if iter.device_type(1) == kHIP #37819". The issue is marked as "Closed" and "New issue". A comment from "mergiam" dated May 5 is visible, containing a "Bug" label and a detailed description of the problem. The description explains that the code accepts tensors of device_type kCPU, kCUDA, or kHIP, but fails to handle kHIP correctly, leading to a segfault. A green callout bubble is overlaid on the code snippet, stating "3 line bugfix took two months to be released!". Below the description, there are sections for "Expected behavior" and "Environment". The "Environment" section lists: PyTorch Version (e.g., 1.0): 1.5, OS (e.g., Linux): CentOS, How you installed PyTorch (conda, pip, source): pip3, and Python version: 3.6. To the right of the issue, there is a pull request interface with a "New pull request" button and a list of pull requests. The pull request list shows a pull request for issue #37819, which was successfully merged and closed the issue. The pull request is labeled "module: rocm" and "triaged".

at::native::copy_impl causes a SegFault if iter.device_type(1) == kHIP #37819 Edit New issue

Closed mergiam opened this issue on May 5 · 1 comment

mergiam commented on May 5

Bug

at::native::copy_impl (<https://github.com/pytorch/pytorch/blob/v1.5.0/aten/src/ATen/native/Copy.cpp#L89>) accepts to process tensors of device_type kCPU, kCUDA, kHIP (see <https://github.com/pytorch/pytorch/blob/v1.5.0/aten/src/ATen/native/Copy.cpp#L79>). To support device to host memcopies, the method checks if the source tensor is located on kCUDA (see <https://github.com/pytorch/pytorch/blob/v1.5.0/aten/src/ATen/native/Copy.cpp#L142>). In this case, it uses copy_kernel_cuda instead of copy_kernel. However, if iter.device_type(1) is kHIP also copy_kernel will be executed, resulting in a segfault, as it would try to access a HIP device pointer from the CPU.

Expected behavior

```
DeviceType device_type = iter.device_type(0);
if (iter.device_type(1) == kCUDA) {
    device_type = kCUDA;
} else if (iter.device_type(1) == kHIP) {
    device_type = kHIP;
}
```

or more generic

```
DeviceType device_type = iter.device_type(0);
if (iter.device_type(1) != kCPU) {
    device_type = iter.device_type(1);
}
```

Environment

- PyTorch Version (e.g., 1.0): 1.5
- OS (e.g., Linux): CentOS
- How you installed PyTorch (conda, pip, source): pip3
- Python version: 3.6

3 line bugfix took two months to be released!

New pull request

Assignee Sort

Assignee Sort

Labels

module: rocm triaged

Projects

None yet

Milestone

No milestone

Linked pull requests

Successfully merging a pull request may close this issue

Issue #37819: Added check for kHIP in ATen/na...

Notifications Customize

Unsubscribe

You're receiving notifications because you were mentioned.

2 participants

May 22

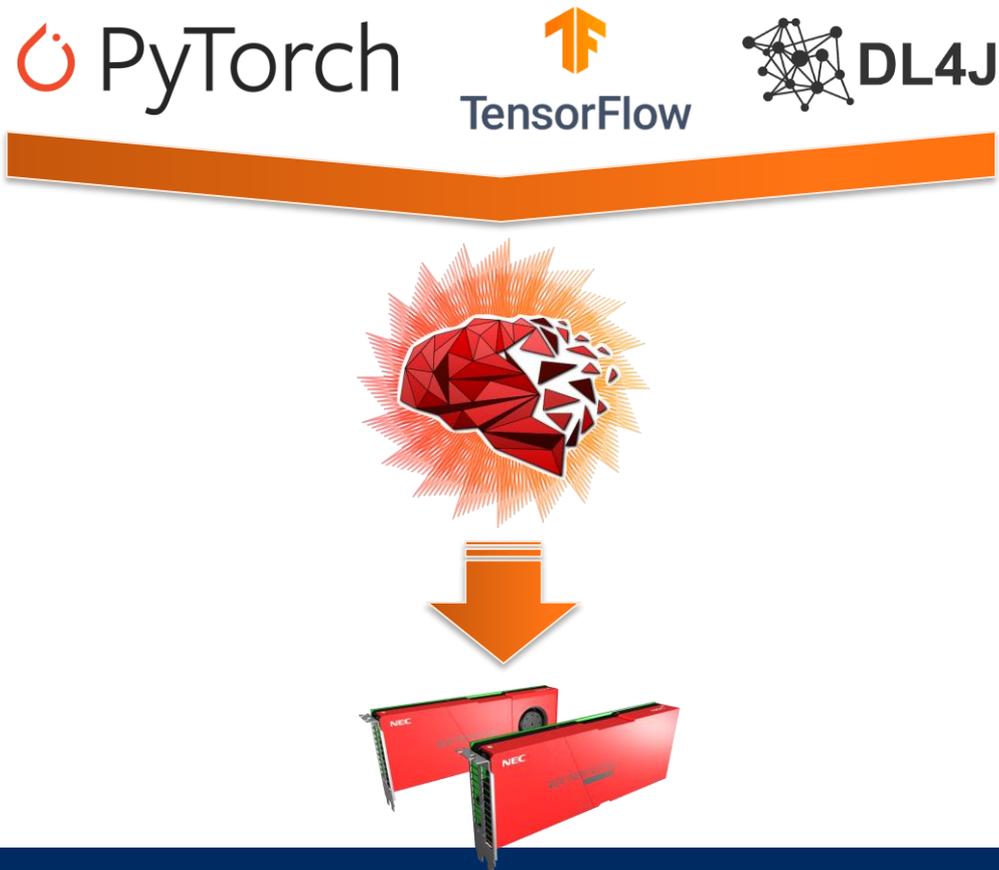
May 27

May 27

The SOL-Project

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)
```

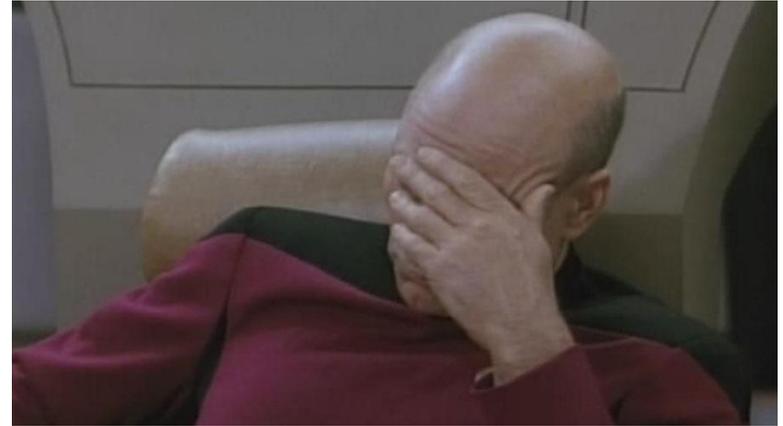
SOL in a nutshell

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):`

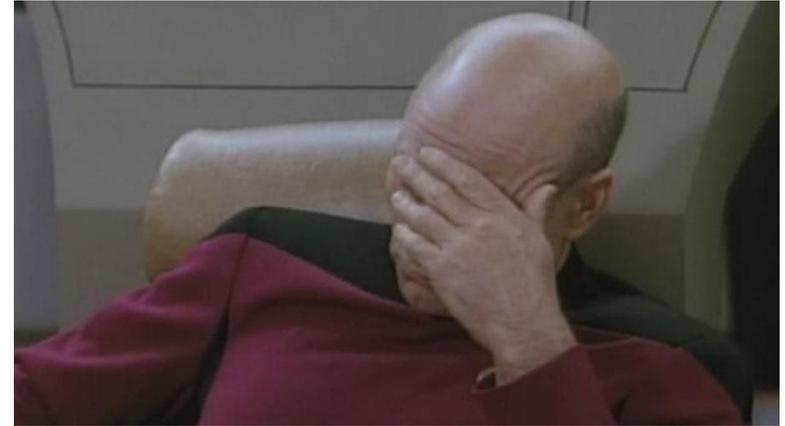
```
for(Batch, OutChannel, Y, X):
    for(InChannel, KernelY, KernelX):
        output[...] += input[...] * weight[...]
        output[...] += bias[...]
```

`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 O0idx = omp_get_thread_num();  
    const int O0 = O0idx / 256;  
    const int O1 = O0idx % 256;  
    float T64[169];  
    #pragma _NEC ivdep  
    for(int O2idx = 0; O2Idx < 169; O2Idx++) {  
        float T63 = 0.0f;  
        for(int I1 = 0; I1 < 512; I1++) {  
            T63 += T61[O0 * 86528 + I1 * 169 + O2idx] * P63_weight[O1 * 512 + I1];  
            T63 = (T63 + P63_bias[O1]);  
            T64[O2Idx] = sol_ncc_max(T63, 0.0f);  
        }  
        T66[O1] = sol_ncc_reduce_add(T64);  
        T66[O1] = (T66[O1] / 169.0f);  
    }  
}
```

The diagram illustrates hardware features with orange arrows pointing to specific parts of the code:

- Cores**: Points to the `omp_get_thread_num()` function call.
- Vector**: Points to the `for(int O2Idx = 0; O2Idx < 169; O2Idx++)` loop.
- inner loop**: Points to the `for(int I1 = 0; I1 < 512; I1++)` loop.
- Reduction**: Points to the `sol_ncc_reduce_add(T64)` function call.

Comments in the code identify operations: `// #1 Convolution: 1x1 Pooling`, `// #1 Convolution: Bias`, and `// #2 ReLU`.

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)

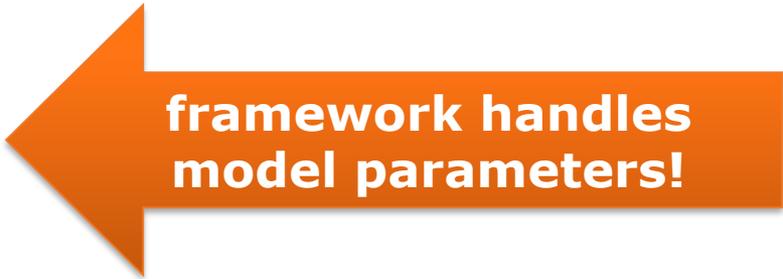
```
import torch
from torchvision import models
import sol.pytorch as sol

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**

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)

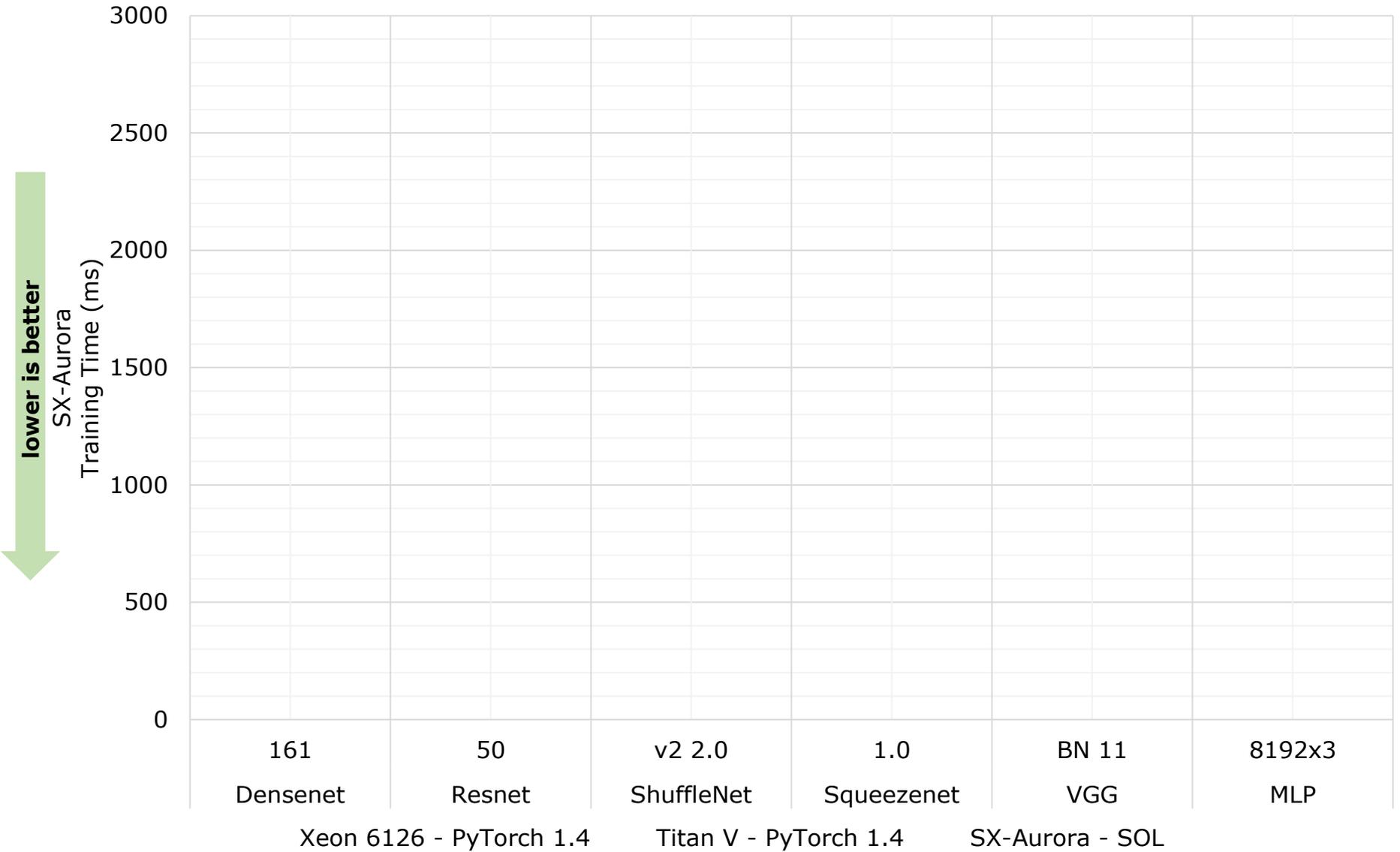
Multi Layer Perceptron (MLP)

Linear/Logistic Regression

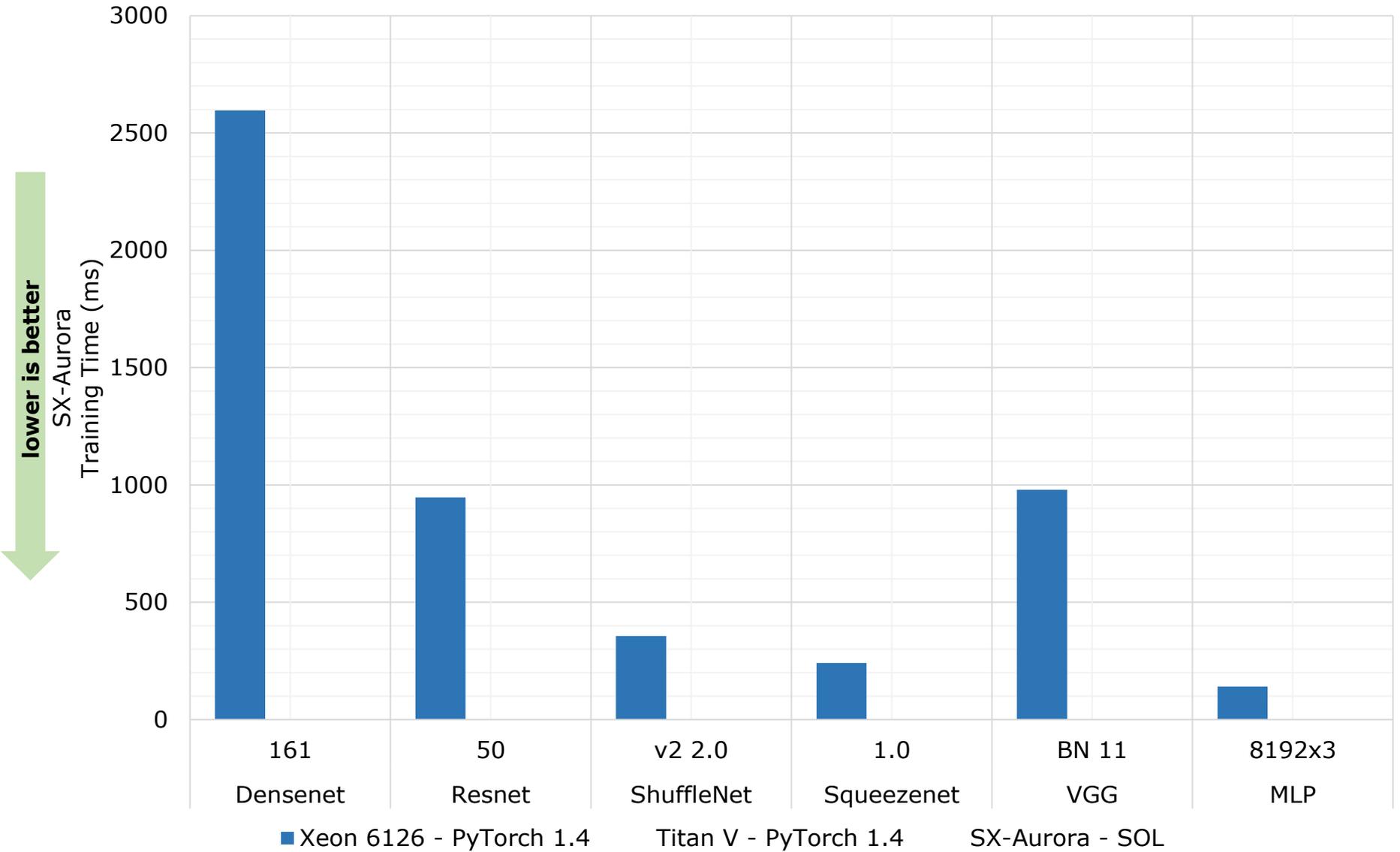
Natural Language Processing

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

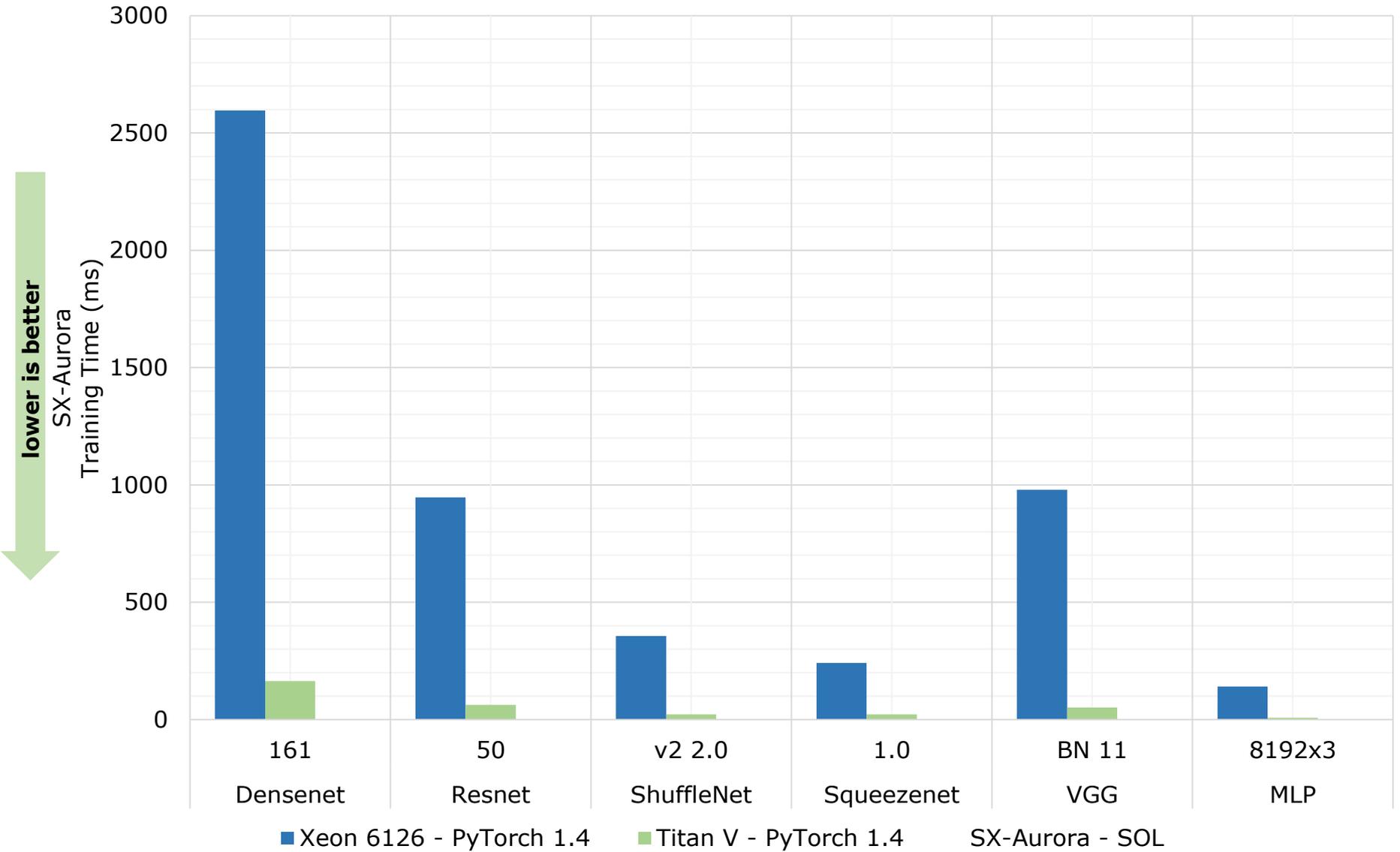
Training Performance (CNN BS=16, MLP BS=64)



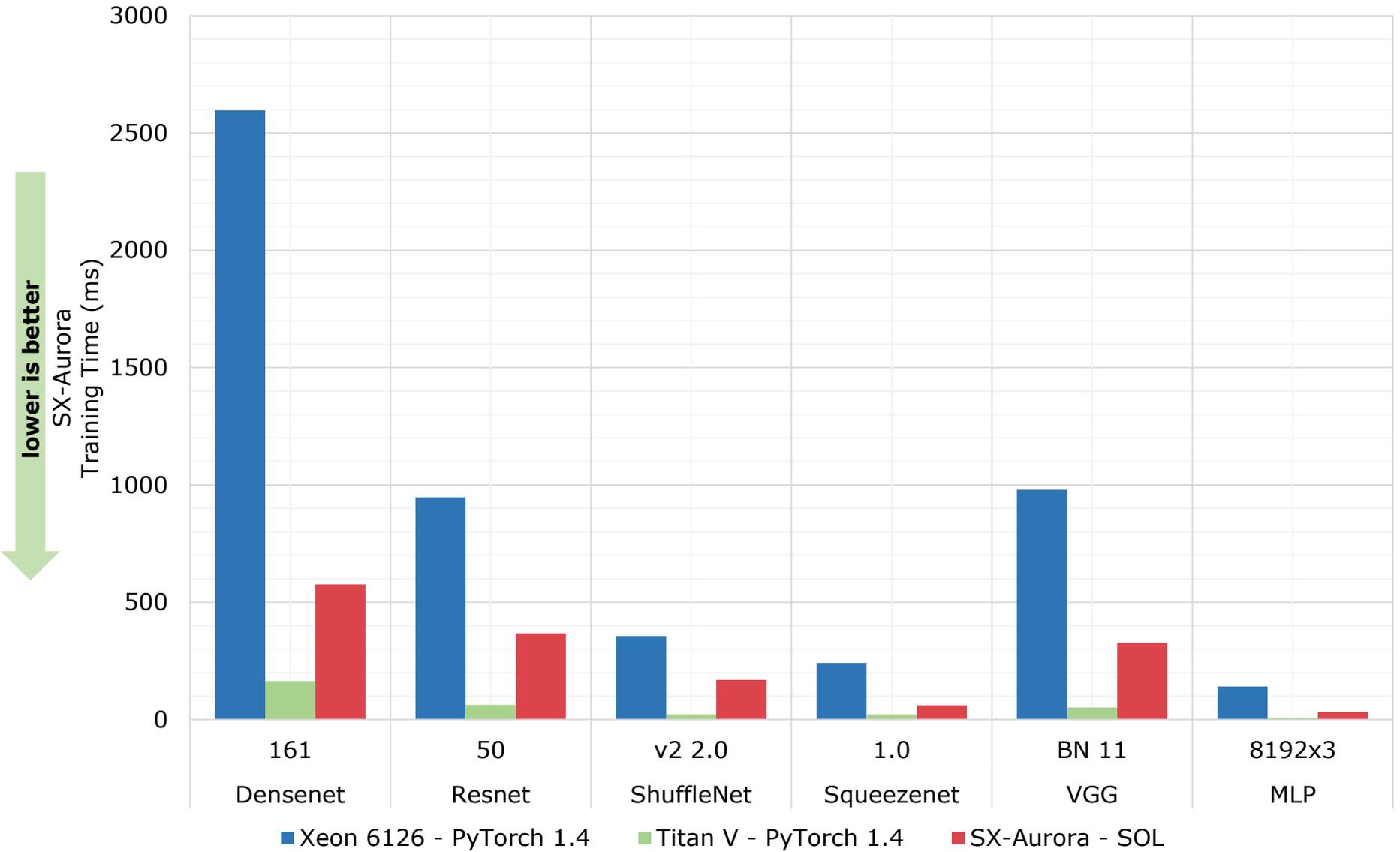
Training Performance (CNN BS=16, MLP BS=64)



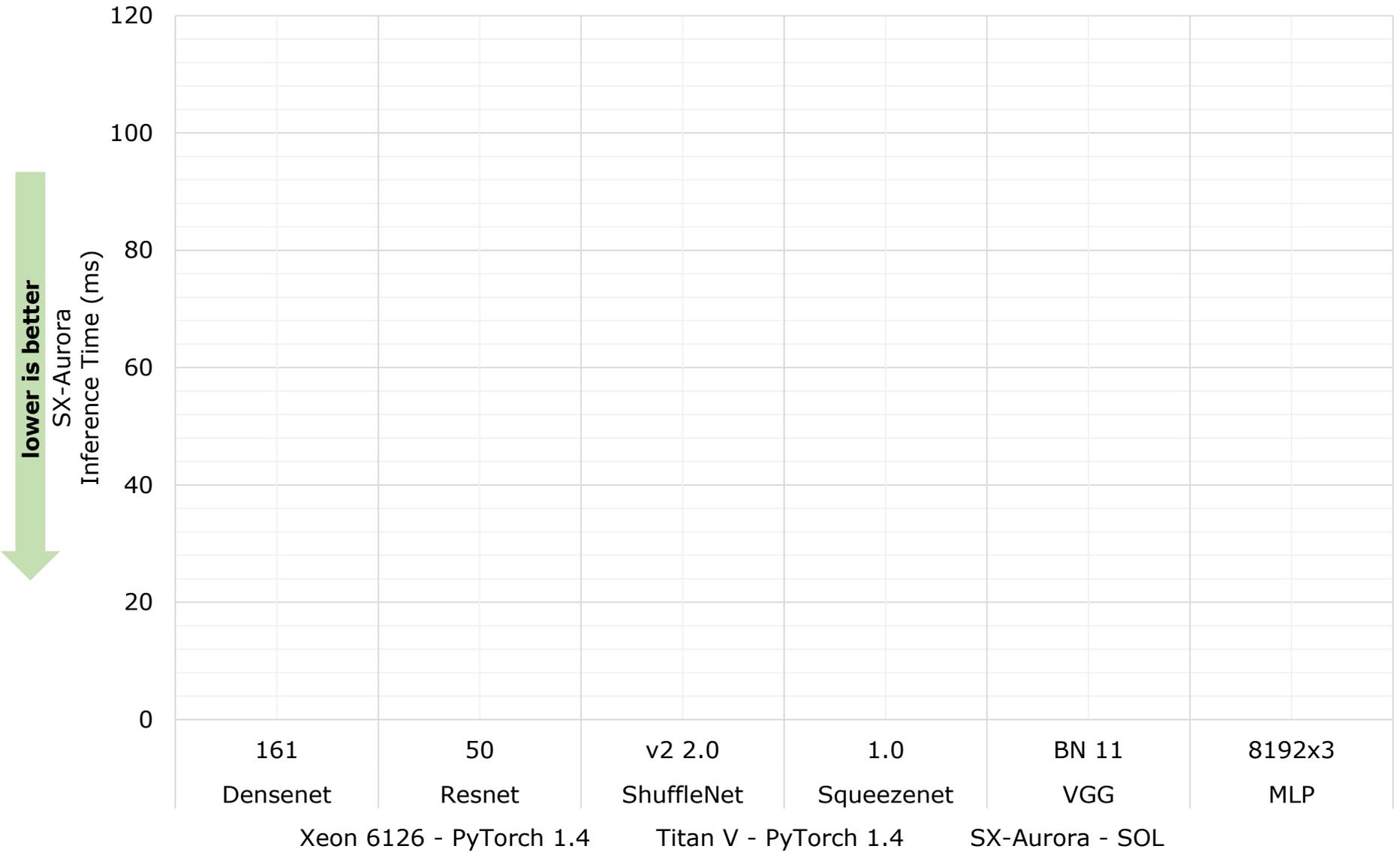
Training Performance (CNN BS=16, MLP BS=64)



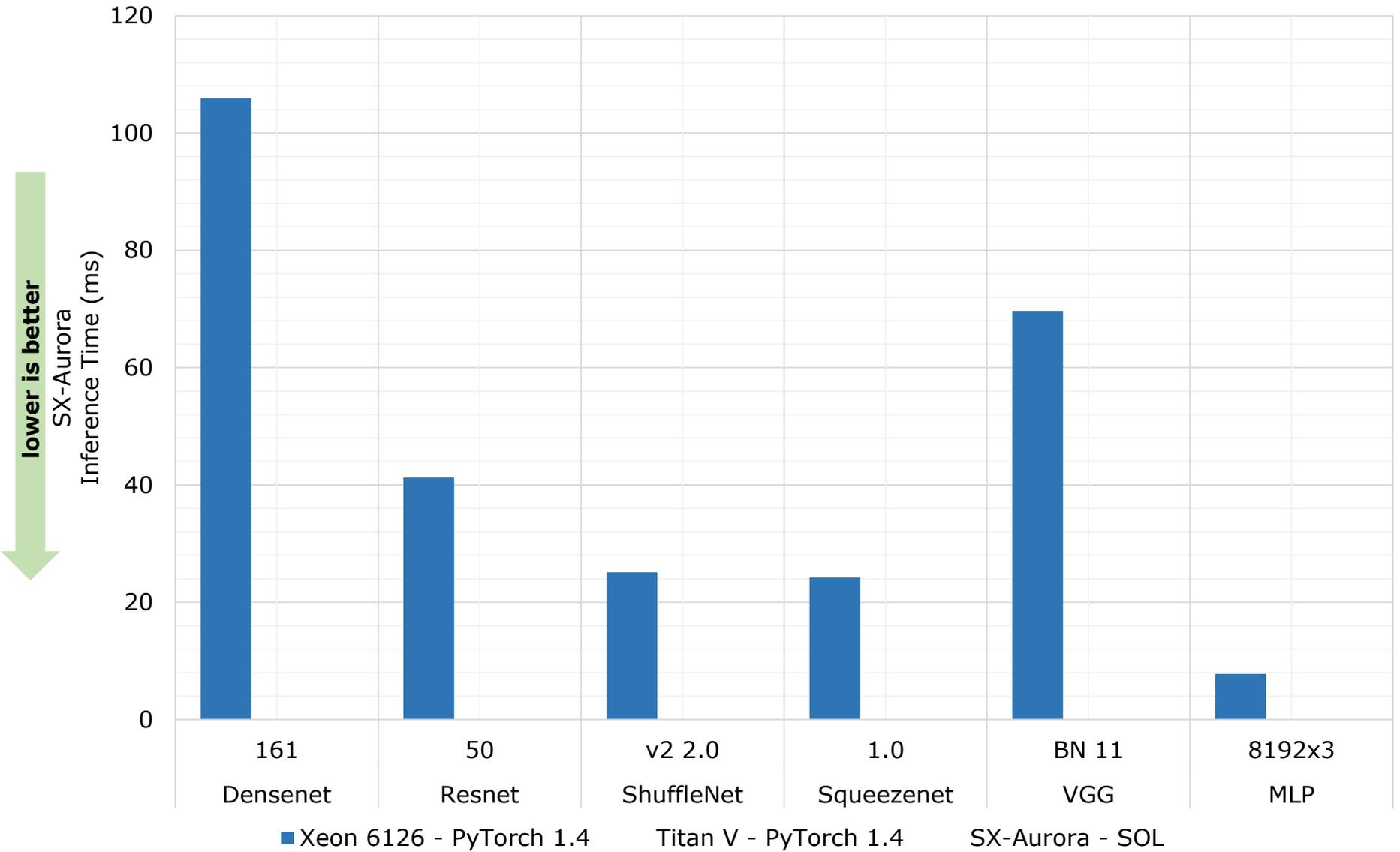
Training Performance (CNN BS=16, MLP BS=64)



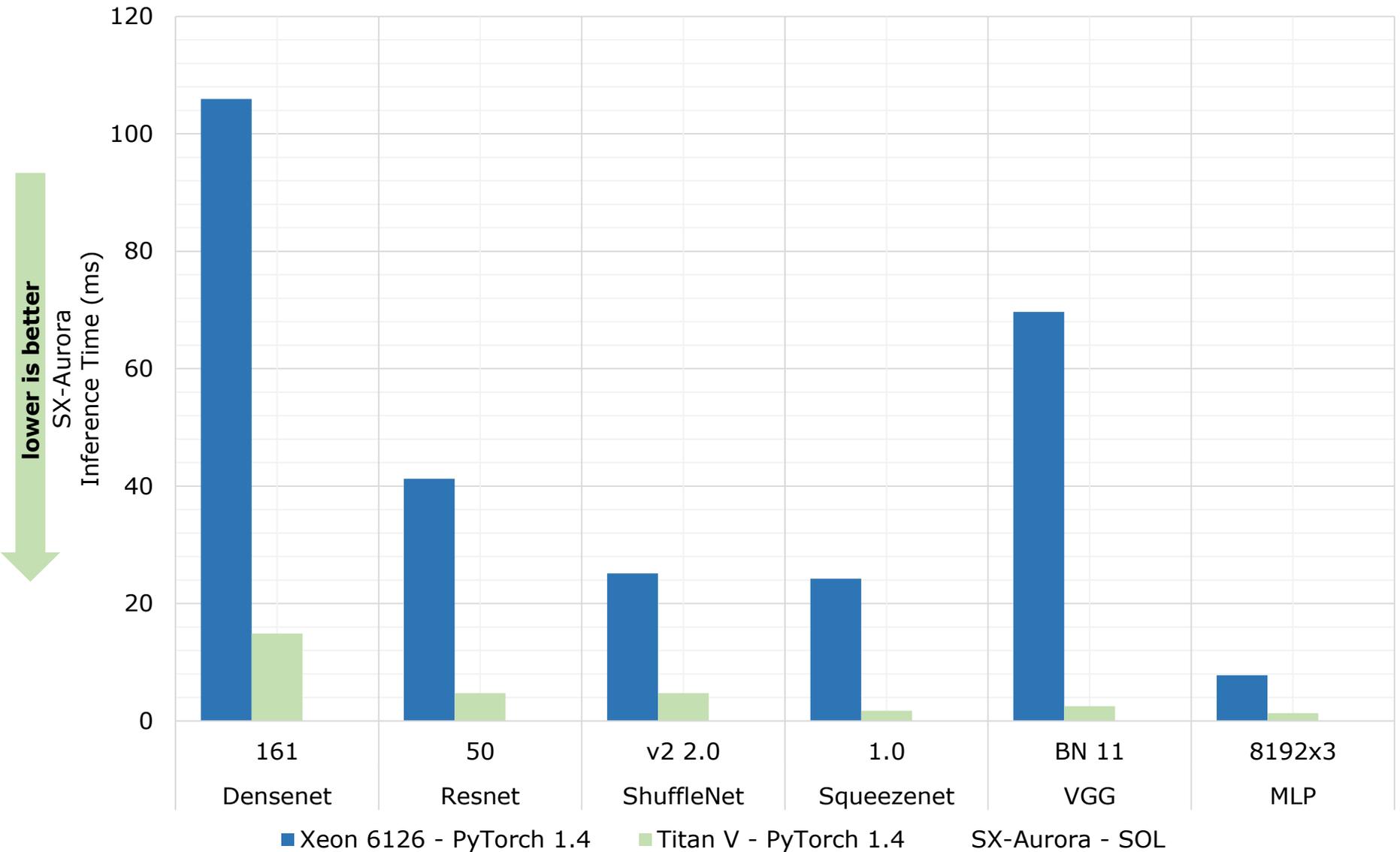
Inference Performance (BS=1)



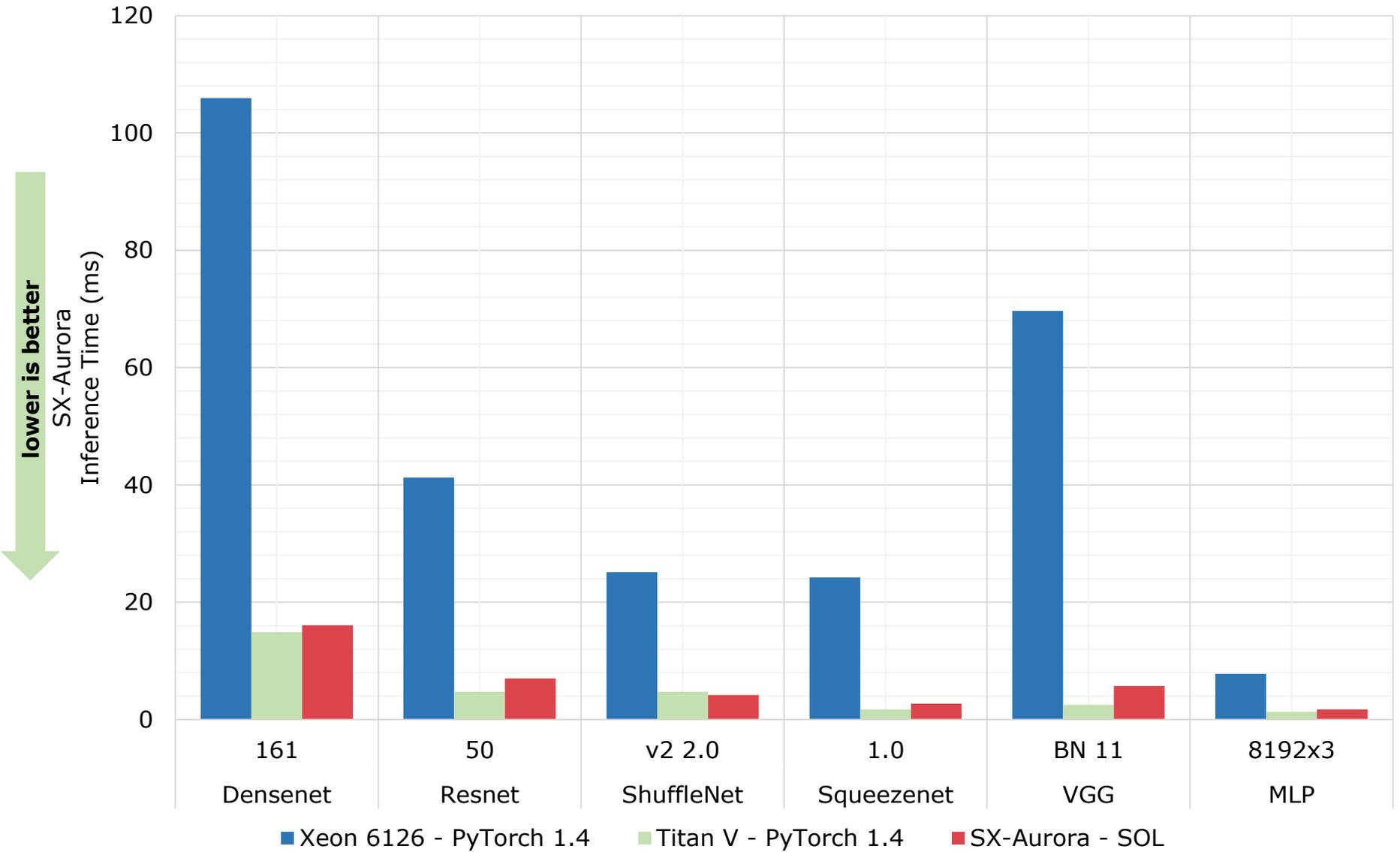
Inference Performance (BS=1)



Inference Performance (BS=1)



Inference Performance (BS=1)



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
- ...

2020:

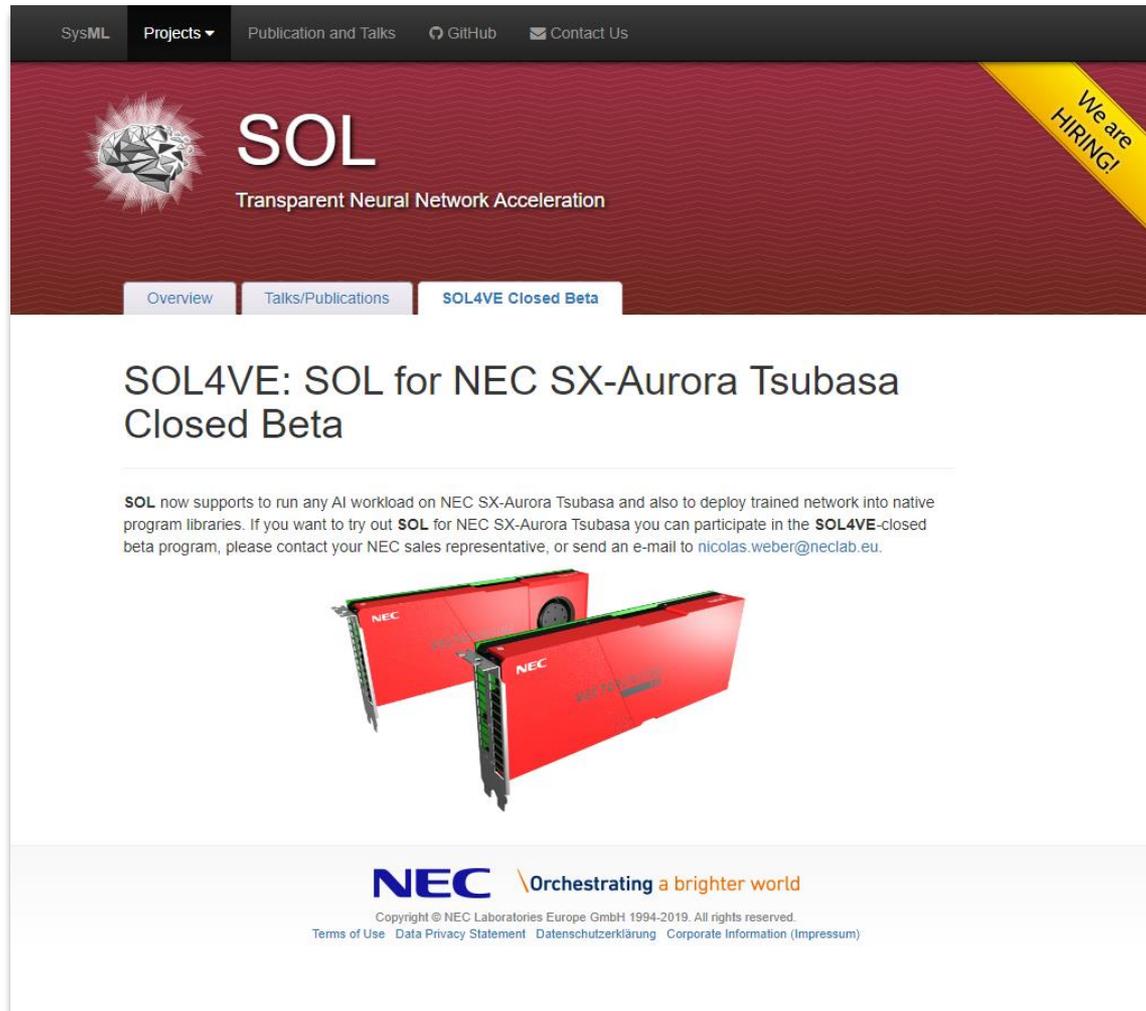
- DL4J (October)
- TensorFlow v2 (December)
- Recurrent Neural Networks (LSTM, GRU)
- torch.nn.DataParallel support for PyTorch

2021:

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

“Can we try SOL4VE?”

Apply for the SOL4VE closed beta via your NEC Sales representative: **www.sol-project.org**



The screenshot shows the website for SOL (Transparent Neural Network Acceleration). The navigation bar includes links for SysML, Projects, Publication and Talks, GitHub, and Contact Us. The main header features the SOL logo and a yellow banner that says "We are HIRING!". Below the header, there are tabs for Overview, Talks/Publications, and SOL4VE Closed Beta. The main content area is titled "SOL4VE: SOL for NEC SX-Aurora Tsubasa Closed Beta" and contains a paragraph explaining that SOL now supports running any AI workload on NEC SX-Aurora Tsubasa and deploying trained networks into native program libraries. It invites users to participate in the SOL4VE-closed beta program by contacting their NEC sales representative or emailing nicolas.weber@neclab.eu. Below the text is an image of two red NEC SX-Aurora Tsubasa accelerator cards. At the bottom, the NEC logo is displayed with the tagline "Orchestrating a brighter world" and copyright information for NEC Laboratories Europe GmbH, 1994-2019.

SysML Projects Publication and Talks GitHub Contact Us

SOL
Transparent Neural Network Acceleration

We are HIRING!

Overview Talks/Publications **SOL4VE Closed Beta**

SOL4VE: SOL for NEC SX-Aurora Tsubasa Closed Beta

SOL now supports to run any AI workload on NEC SX-Aurora Tsubasa and also to deploy trained network into native program libraries. If you want to try out SOL for NEC SX-Aurora Tsubasa you can participate in the SOL4VE-closed beta program, please contact your NEC sales representative, or send an e-mail to nicolas.weber@neclab.eu.



NEC | Orchestrating a brighter world

Copyright © NEC Laboratories Europe GmbH 1994-2019. All rights reserved.
Terms of Use Data Privacy Statement Datenschutzerklärung Corporate Information (Impressum)

Dr. Nicolas Weber

Intelligent Software Systems Group

Senior Software Engineer

NEC Laboratories Europe

nicolas.weber@neclab.eu

www.sol-project.org

