

Orchestrating a brighter world

NEC



Integration of NEC SX-Aurora into AI Frameworks (A **SOL** story)

Dr. Nicolas Weber (NEC Labs Europe)

Motivation

Obvious: Everyone does AI today!

- AI-optimized Fridges, Microwaves, Toasters, T-800 Terminators, ...

But where to start?



TensorFlow



Chainer

theano



Caffe2



PyTorch

Integration into existing frameworks is expensive

Each framework has its own APIs

- Appr
very

Filters Labels 171 Milestones 1 [New pull request](#)

☒ Clear current search query, filters, and sorts

0 Open	32 Closed	Author	Labels	Projects	Milestones	Reviews	Assignee	Sort
[Caffe2] Enable AMD/MIOpen ops for Caffe2 open source	3							
#8306 by petrex was merged on Jun 13, 2018 • Approved								
PyTorch/Caffe2 Build Path Changes (ROCm path) + MIOpen Integration open source	18							
#8257 by Jorghi12 was closed on Sep 21, 2018								
[Caffe2] Enabling AMD GPU Backend for Caffe2 open source	32							
#7566 by petrex was merged on May 24, 2018 • Approved								
[Caffe2] Support non peer access in muji and fix bug when reduced_affix is empty caffe2 open source	22							
#6896 by daquexian was merged on Jun 4, 2018 • Approved								
PyTorch AMD Build Scripts open source	68							
#6625 by Jorghi12 was merged on May 16, 2018 • Approved								
Initial commit for PyTorch to run on AMD hardware using the ROCM software stack open source	4							
#3544 by wsttiger was closed on Nov 9, 2017 • Changes requested								
[WIP] Initial HIP-ification of PyTorch code for AMD hardware open source	5							
#2365 by wsttiger was closed on Sep 18, 2017								

[Previous](#) [1](#) [2](#) [Next](#)

The SOL-Project

SOL is a full stack AI acceleration middleware

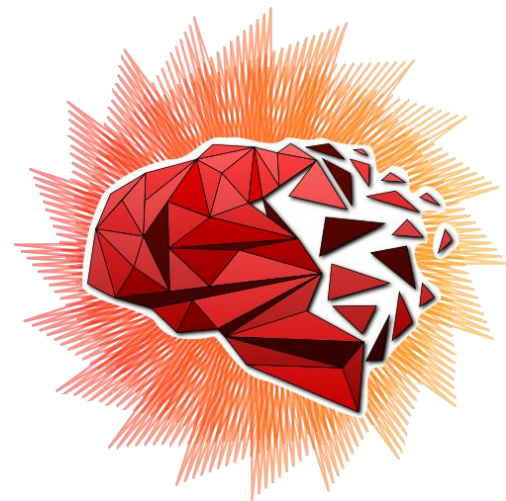
- Optimizations range from mathematical/algorithmic down to actual implementations/code generation
- Add-on to AI frameworks that does not require any code changes

Tightly integrates into existing frameworks

- PyTorch
- TensorFlow
- MxNet (in development)

Broad support for hardware architectures

- X86 CPUs
- NVIDIA GPUs
- ARM64 CPUs
- **NEC SX-Aurora Tsubasa**



SOL in a nutshell

What data scientists see:

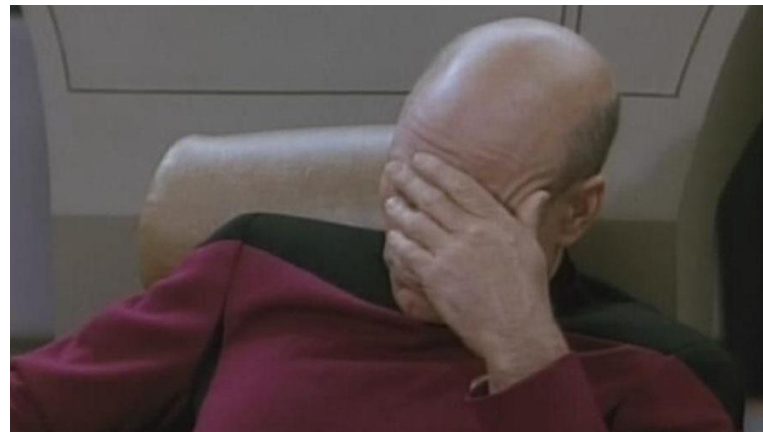
```
x = Conv(x, kernel=1x1, bias=True)
x = ReLU(x)
x = AvgPooling(x, kernel=13x13)
```

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 = blockIdx.x;  
    const int O0 = O0idx / 256;  
    const int O1 = O0idx % 256;  
    __shared__ float T64[169];  
    for(int O2idx = threadIdx.x; O2idx < 169; O2idx += 128) {  
        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] = fmaxf(T63, 0.0f);  
    }  
    T66[O1] = REDUCE_ADD(T64);  
    T66[O1] = (T66[O1] / 169.0f);  
}
```

CUDA blocks

shared memory

CUDA cores

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
import sol.pytorch as sol
```

```
py_model = models.__dict__["..."]()
input = torch.rand(1, 32, 224, 224)
sol_model = sol.optimize(py_model, input.size())
sol_model.load_state_dict(py_model.state_dict())
sol.device.set(sol.device.dev, 0)
output = sol_model(input)
```

How to integrates SOL into the frameworks?

- SOL injects its optimized code as Custom Layer into the framework

```
class SolLayer(torch.nn.Module):
```

```
    def __init__(self):  
        self.ParamA = ...  
        self.ParamB = ...
```



**framework handles
model parameters!**

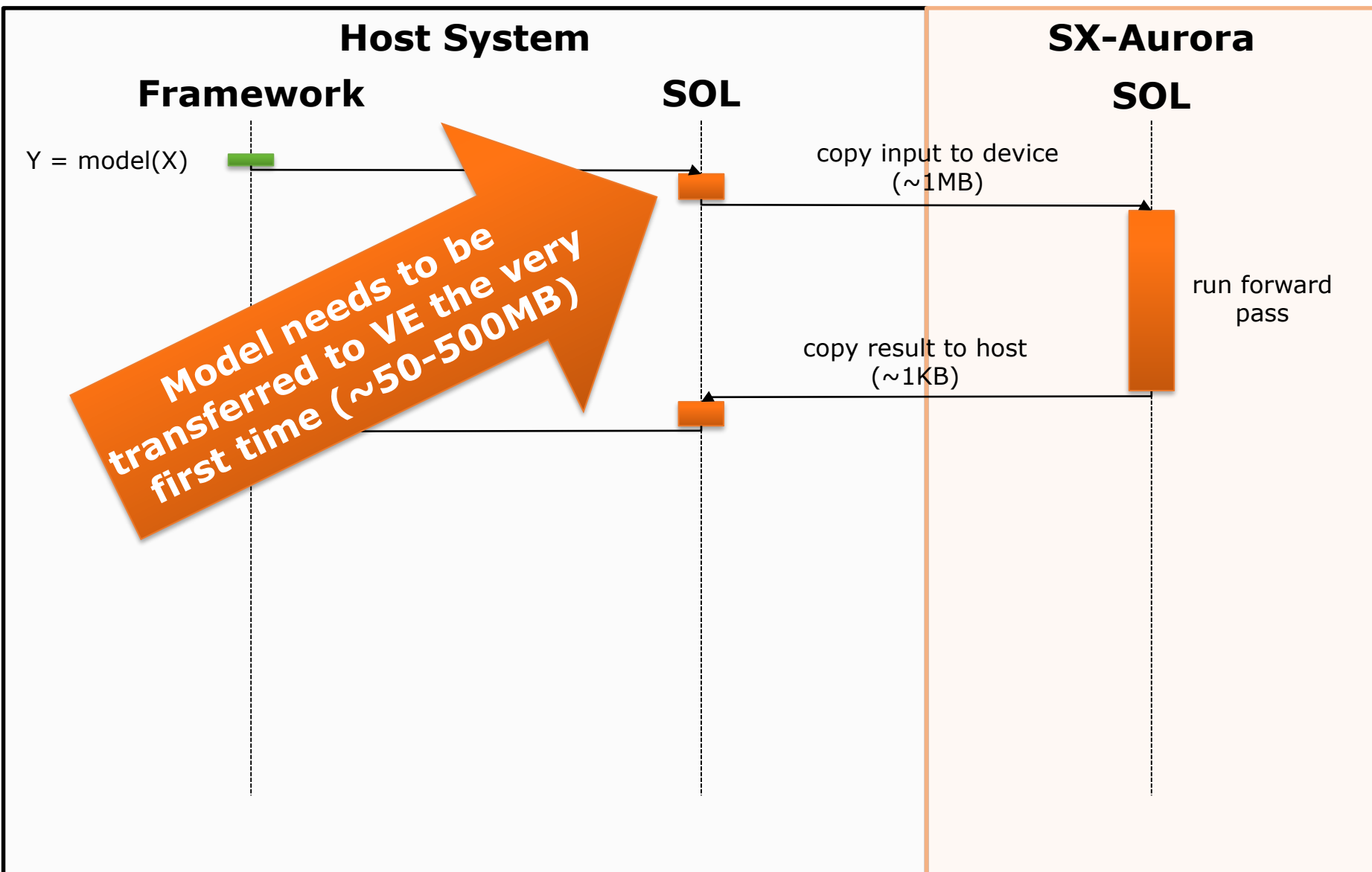
```
    def forward(self, X):
```

```
        return sol.run(X, self.ParamA, self.ParamB)
```

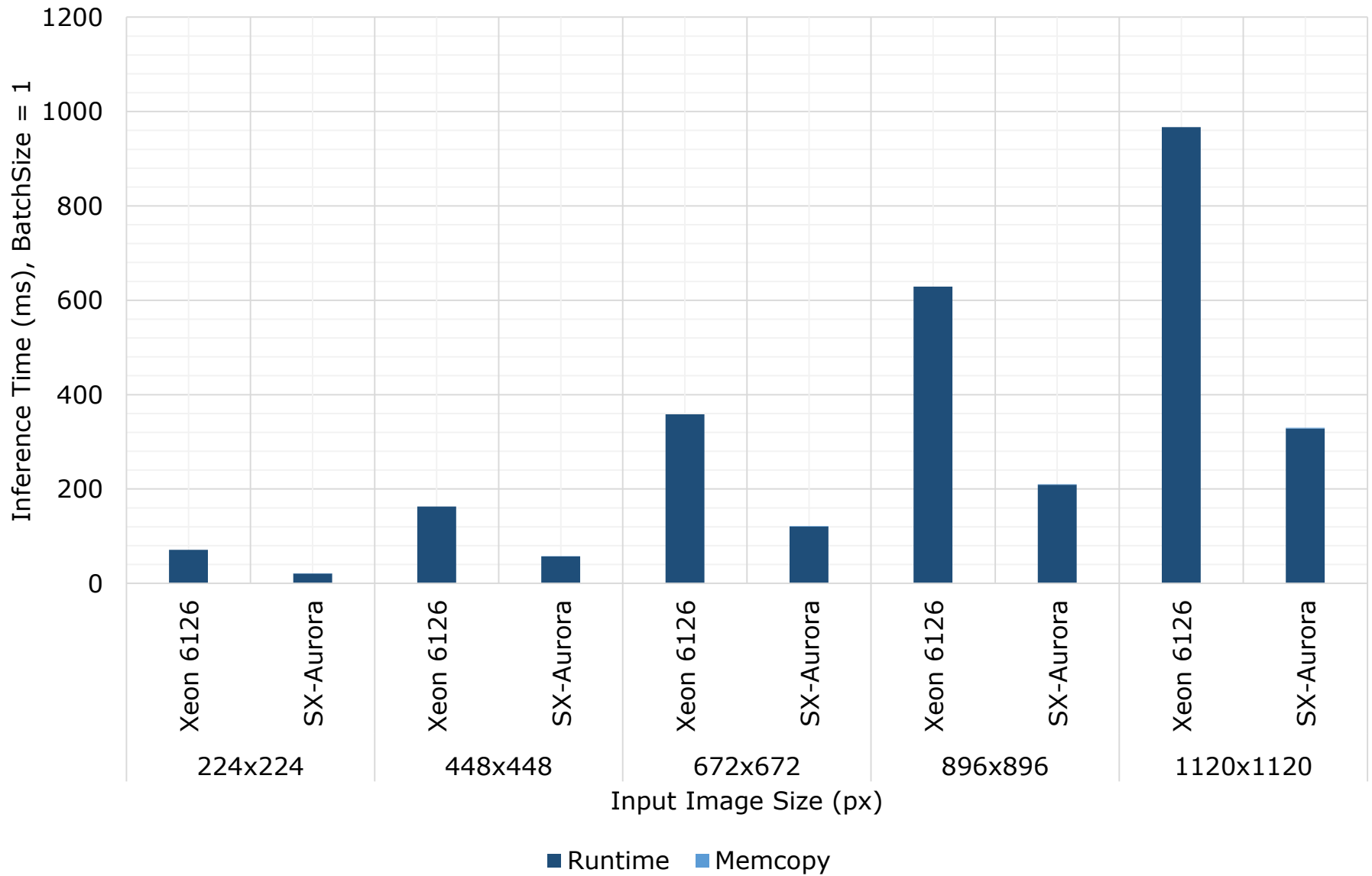


**SOL handles
execution**

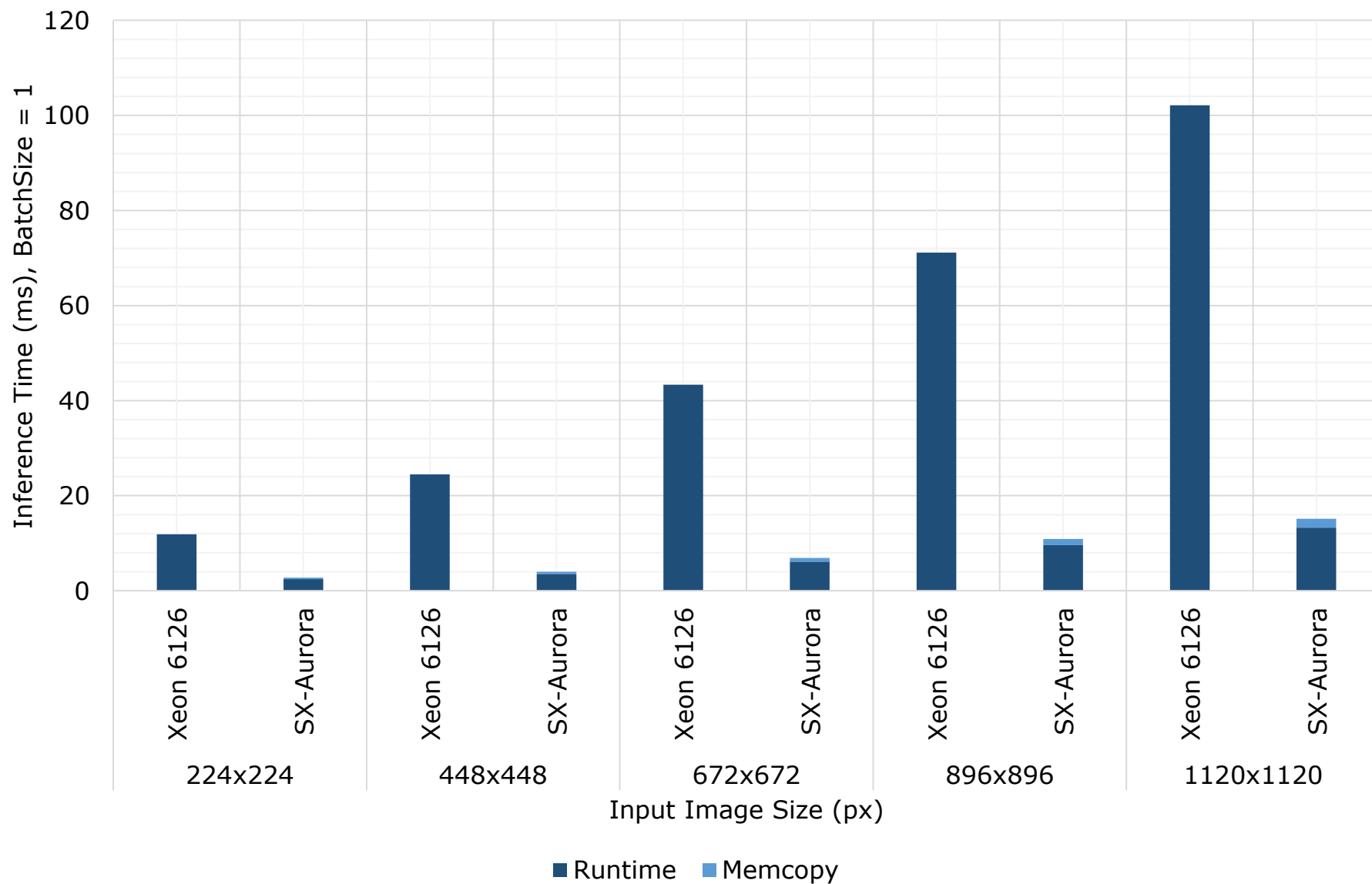
SOL SX-Aurora Inference



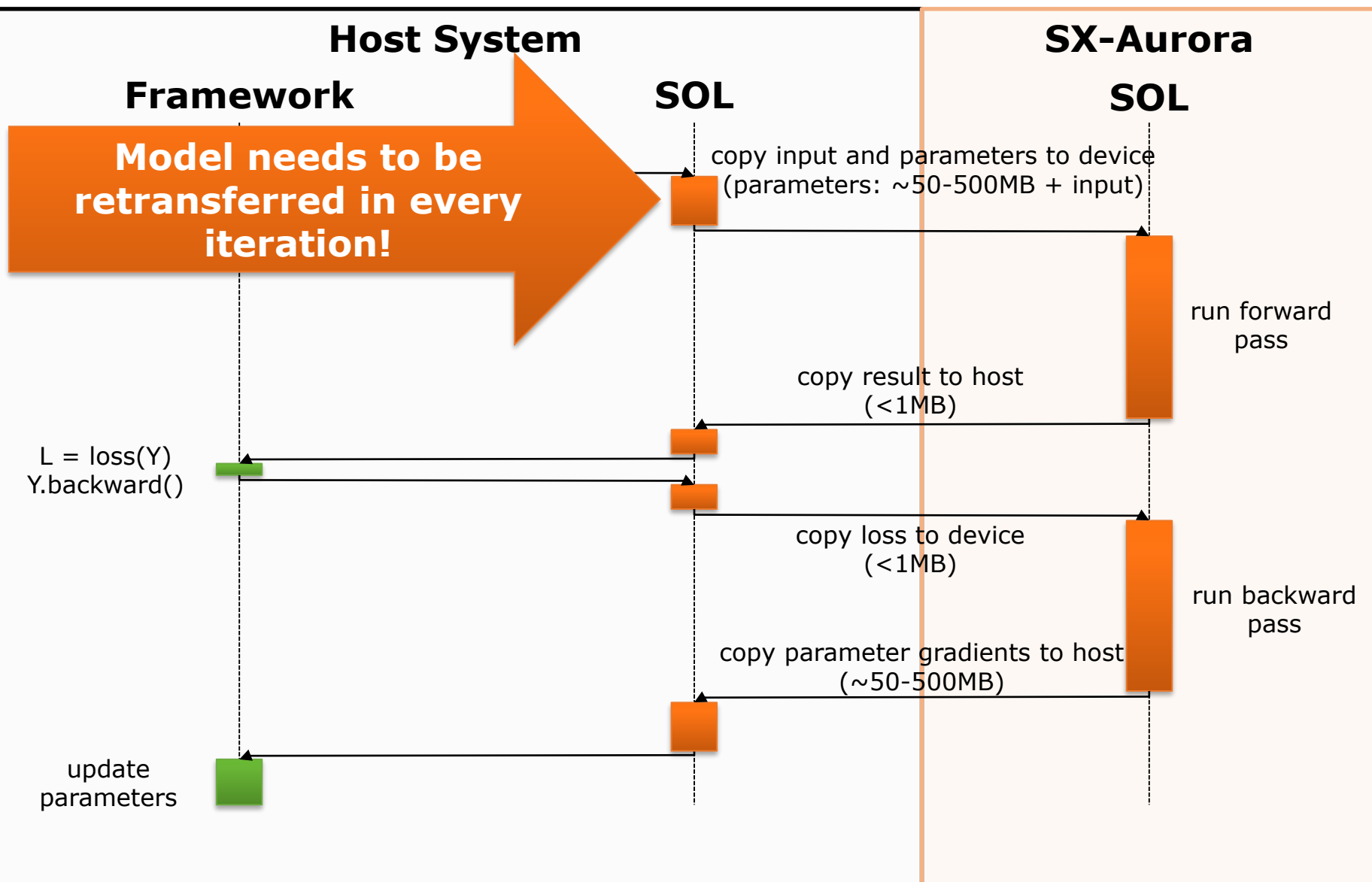
Inference on DenseNet 121



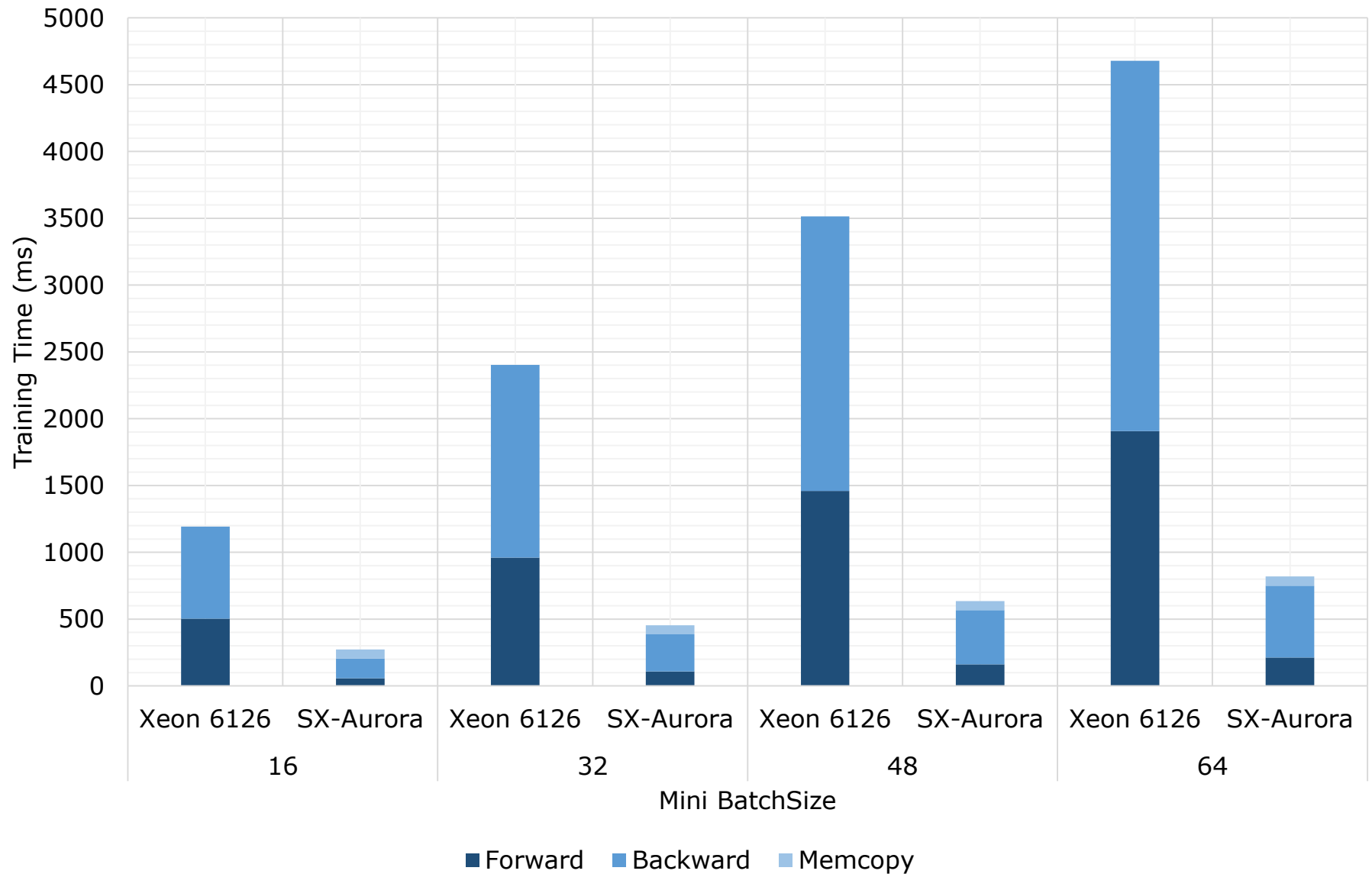
Inference ShuffleNet v2/0.5



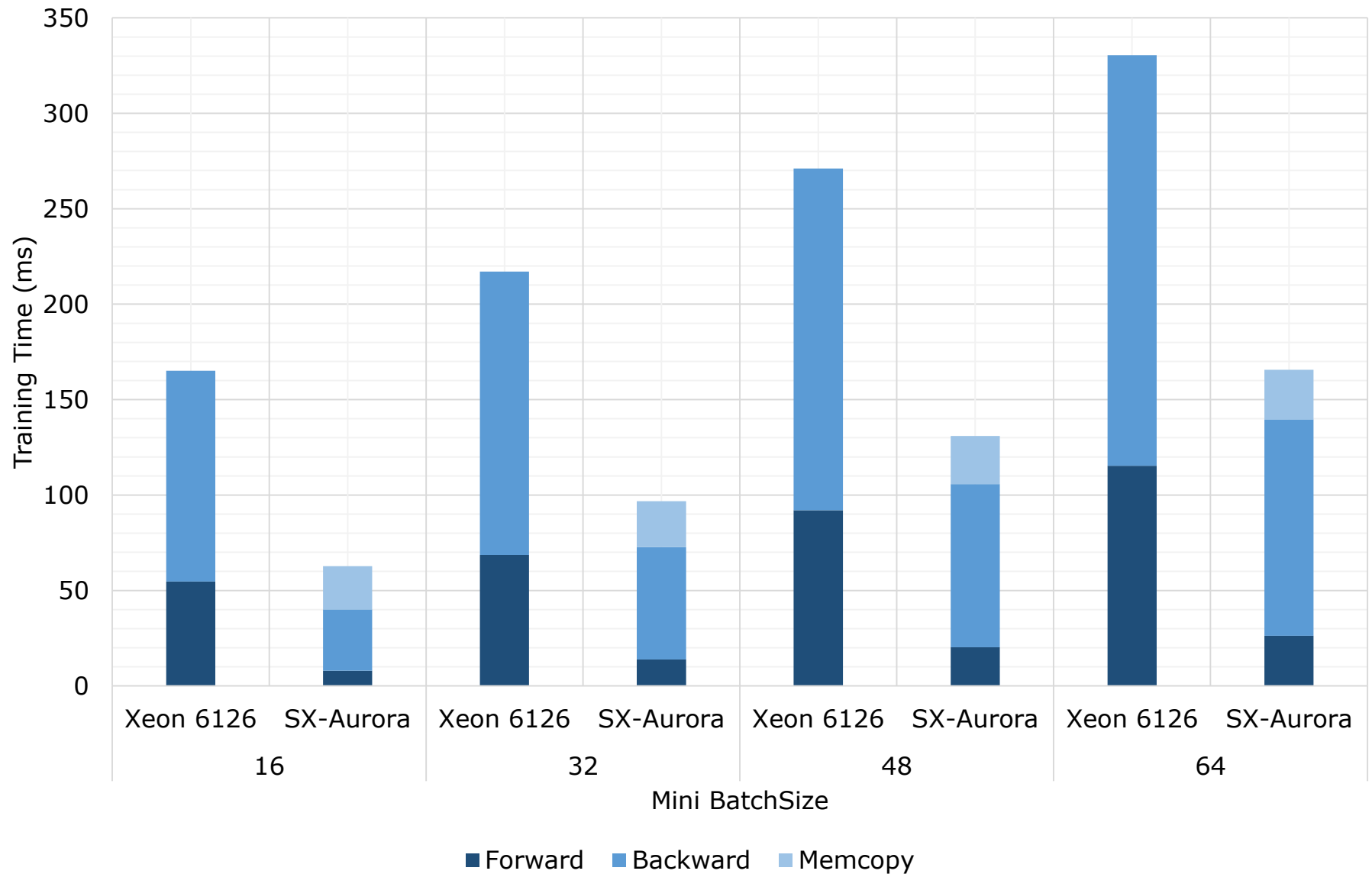
SOL SX-Aurora Training



Training on DenseNet 121



Training on ShuffleNet v2/0.5



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

“How can the neural network be used in an application?”

■ SOL supports model deployment!

```
sol.deploy(trained_model, [1, 3, 224, 224],  
target=sol.deployment.SharedLib, 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
```

“How can I add new functionality, not supported by the framework?”

```
class MyLayer(torch.nn.Layer):  
    def __init__(self, ...):  
        super().__init__()  
  
        self.ParamA = torch.nn.Parameter(...)  
        self.ParamB = torch.nn.Parameter(...)  
  
    def forward(self, X):  
        # ... code that executes when PyTorch  
        # executes the layer ...
```

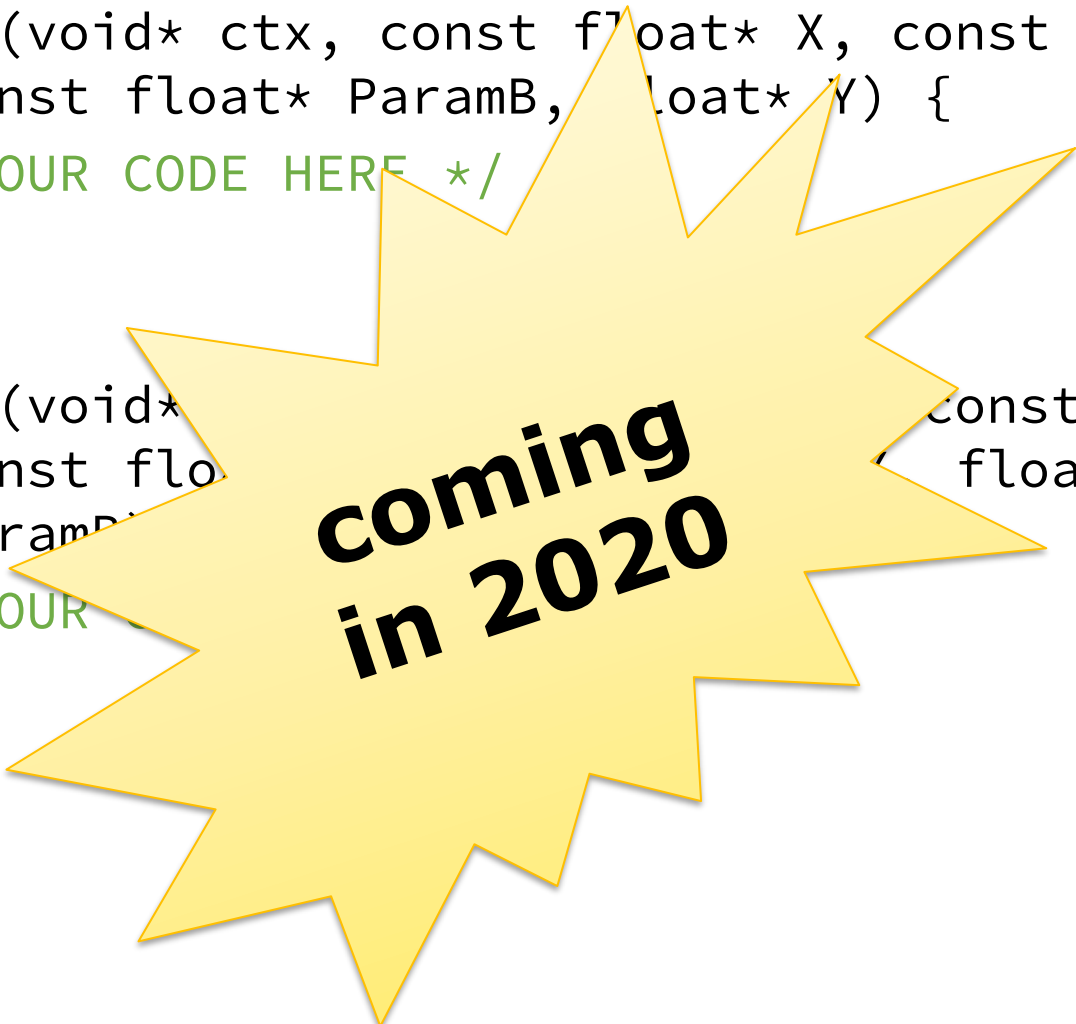
“How can I add new functionality, not supported by the framework?”

```
class MyLayer(sol.nn.CustomLayer):  
    def __init__(self, ...):  
        super().__init__({  
            sol.device.nvidia: ["libMyCUDA.so", "FwdCUDA",  
"BwdCUDA"],  
            sol.device.ve:      ["libMyVE.so", "FwdVE", "BwdVE"]  
        })  
        self.ParamA = torch.nn.Parameter(...)  
        self.ParamB = torch.nn.Parameter(...)  
  
    def forward(self, X):  
        # ... code that executes when PyTorch  
        # executes the layer ...
```

“How can I add new functionality, not supported by the framework?”

```
void FwdVE(void* ctx, const float* X, const float* ParamA, const float* ParamB, float* Y) {  
    /* YOUR CODE HERE */  
}
```

```
void BwdVE(void* ctx, const float* ParamA, const float* dParamA, float* dParamB)  
    /* YOUR CODE HERE */  
}
```



**coming
in 2020**

SOL Development Agenda 2020

Natural Language Processing:

- Recurrent Neural Networks (RNN)
- Transformers

Devices:

- AMD GPUs
- ARM Mali GPUs
- Intel Movidius Myriad

Frameworks:

- TensorFlow 2.0
- MxNet

Features:

- Custom Layer Support

...

“Can we try SOL4VE?”

Apply for the closed SOL4VE beta on:
www.sol-project.org



SOL
Transparent Neural Network Acceleration

We are HIRING!

or talk to any of these two handsome guys after the presentation

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 \(Impressum\)](#)

FROM

- Sparse
- Matrix
- Machine
- Operations
- https

 takuya-araki renamed to knn_unsupervised.cc

..	
agglomerative	corrected program option parameter (NTI)
dataframe_tools	correct to remove CXXFLAGS in LINK, which contains options that canno...
dbscan	add dbscan sample
document_matrix	correct to remove CXXFLAGS in LINK, which contains options that canno...
eigen	correct to remove CXXFLAGS in LINK, which contains options that canno...
factorization_machine	correct to remove CXXFLAGS in LINK, which contains options that canno...
forest	update: simplify tree samples
fp_growth	add fp_growth sample
gbts	update: simplify tree samples
glm	correct to remove CXXFLAGS in LINK, which contains options that canno...
graph	modified interface etc.
kmeans	correct to remove CXXFLAGS in LINK, which contains options that canno...
knn	renamed to knn_unsupervised.cc
lda	add samples for lda
matrix_tools	convert mtx filt to crs
matrixfactorization	correct to remove CXXFLAGS in LINK, which contains options that canno...
pca	add pca sample
spectral	add spectral clustering sample
svd	correct to remove CXXFLAGS in LINK, which contains options that canno...
tree	update: simplify tree samples
wikipedia2matrix	correct to remove CXXFLAGS in LINK, which contains options that canno...





Dr. Nicolas Weber

Intelligent Software Systems Group
Senior Researcher

NEC Laboratories Europe

nicolas.weber@neclab.eu

 **www.sol-project.org**

