

Orchestrating a brighter world



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

Dr. Nicolas Weber (NEC Labs Europe)

Obvious: Everyone does AI today!

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



Integration into existing frameworks is expensive

Each framework has its own APIs

- Approach very

Any plan

Closed Arjuna

Arjuna

Q

Do you

dliben

Our IR
When
builder
Eventually
lowerin

BE Smart

The screenshot shows a GitHub search results page with the query "amd hip". The results list several pull requests:

- [Caffe2] Enable AMD/MIOOPEN ops for Caffe2 (open source) - Merged on Jun 13, 2018
- PyTorch/Caffe2 Build Path Changes (ROCM path) + MIOpen Integration (open source) - Closed on Sep 21, 2018
- [Caffe2] Enabling AMD GPU Backend for Caffe2 (open source) - Merged on May 24, 2018
- [Caffe2] Support non peer access in muji and fix bug when reduced_affix is empty (caffe2 open source) - Merged on Jun 4, 2018
- PyTorch AMD Build Scripts (open source) - Merged on May 16, 2018
- Initial commit for PyTorch to run on AMD hardware using the ROCM software stack (open source) - Closed on Nov 9, 2017
- [WIP] Initial HIP-ification of PyTorch code for AMD hardware (open source) - Closed on Sep 18, 2017

On the right side of the slide, there are two vertical timelines. The top timeline shows a message from Arjuna dated May 22: "Great synergy with the". The bottom timeline shows a message from dliben dated May 27: "LIR into Glow back, and target any".

The SOL-Project

SOL is a full stack AI acceleration middleware

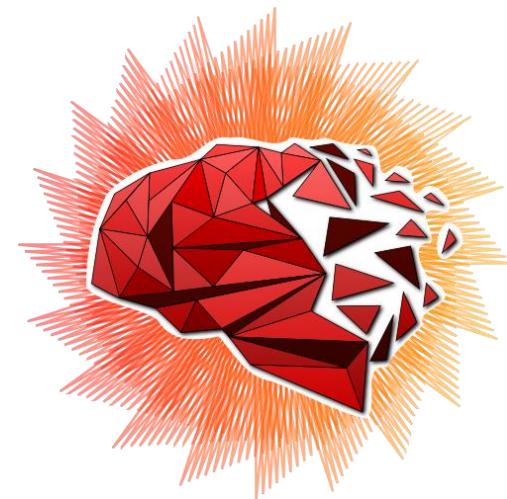
- Optimizations range from mathematical/algorithmic down to actual implementations/code generation
- Add-on to AI frameworks that does not require any code changes

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:

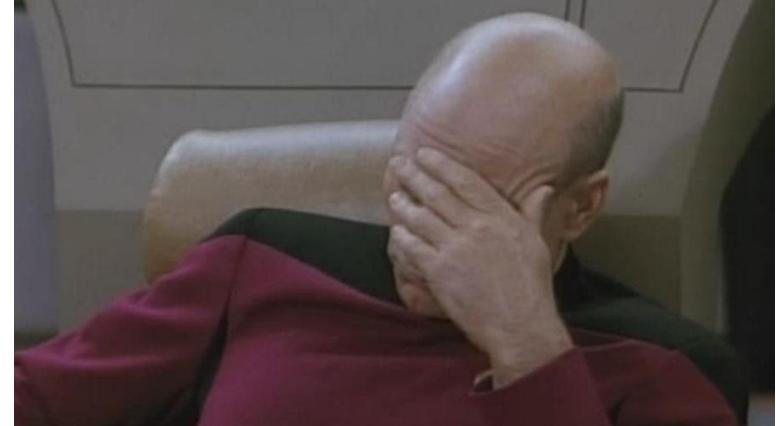
```
function(Conv):  
    for(Batch, OutChannel, Y, X):  
        for(InChannel, KernelY, KernelX):  
            output[...] += input[...] * weight[...]  
            output[...] += bias[...]
```

```
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)

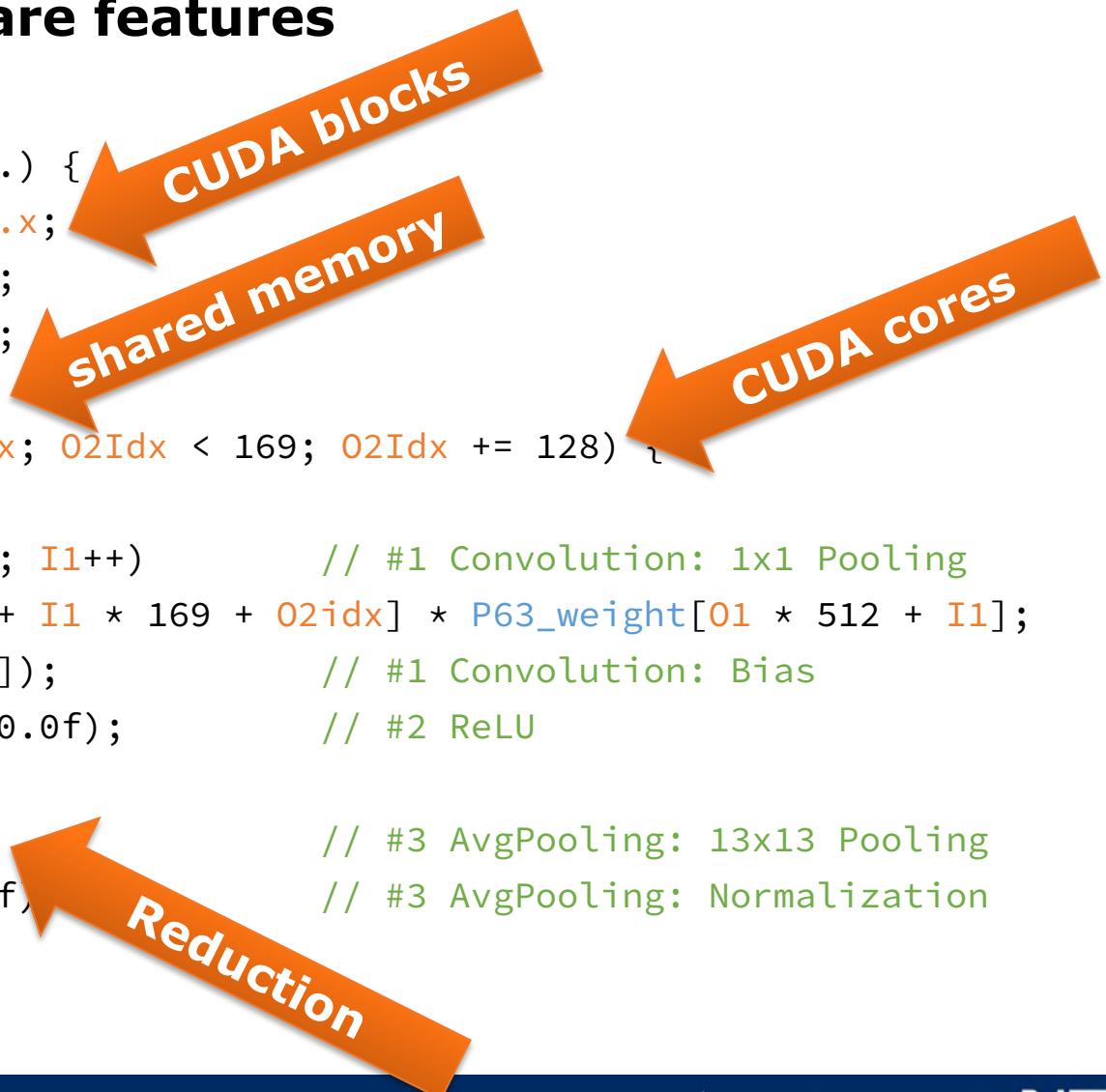
What we actually want:

```
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 00idx = blockIdx.x;  
    const int 00 = 00idx / 256;  
    const int 01 = 00idx % 256;  
    __shared__ float T64[169];  
    for(int 02Idx = threadIdx.x; 02Idx < 169; 02Idx += 128) {  
        float T63 = 0.0f;  
        for(int I1 = 0; I1 < 512; I1++) // #1 Convolution: 1x1 Pooling  
            T63 += T61[00 * 86528 + I1 * 169 + 02Idx] * P63_weight[01 * 512 + I1];  
        T63 = (T63 + P63_bias[01]); // #1 Convolution: Bias  
        T64[02Idx] = fmaxf(T63, 0.0f); // #2 ReLU  
    }  
    T66[01] = REDUCE_ADD(T64); // #3 AvgPooling: 13x13 Pooling  
    T66[01] = (T66[01] / 169.0f); // #3 AvgPooling: Normalization  
}
```



SOL Usage (Pytorch)

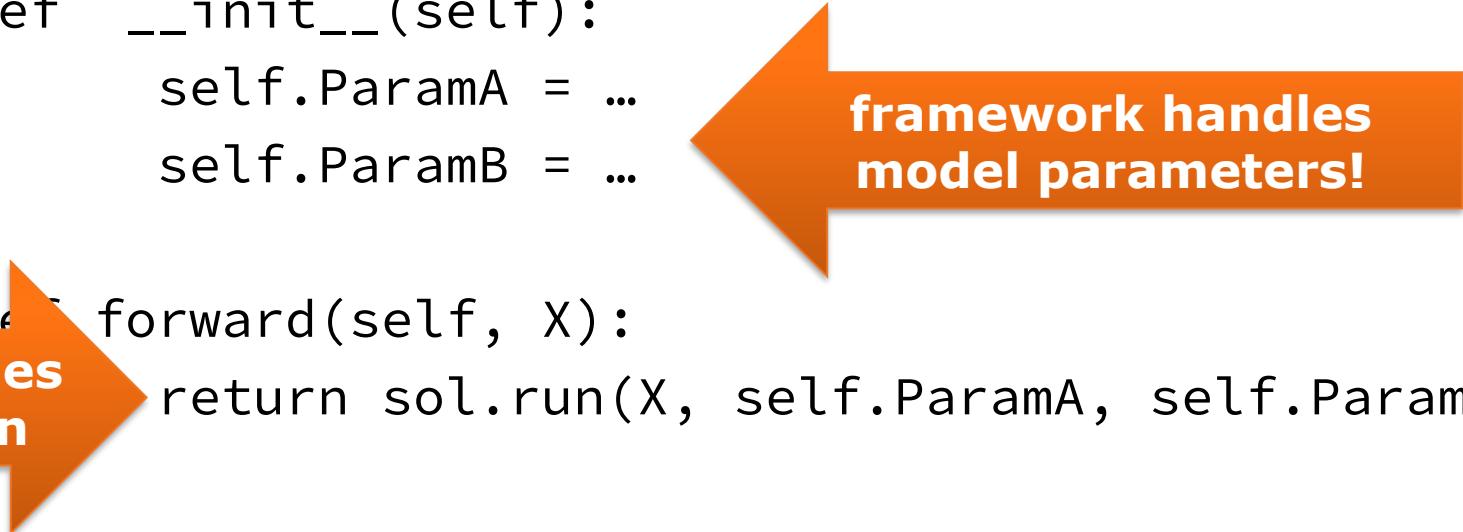
```
import torch
from torchvision import models
import sol.pytorch as sol

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.vc, 0)
output = sol_model(input)
```

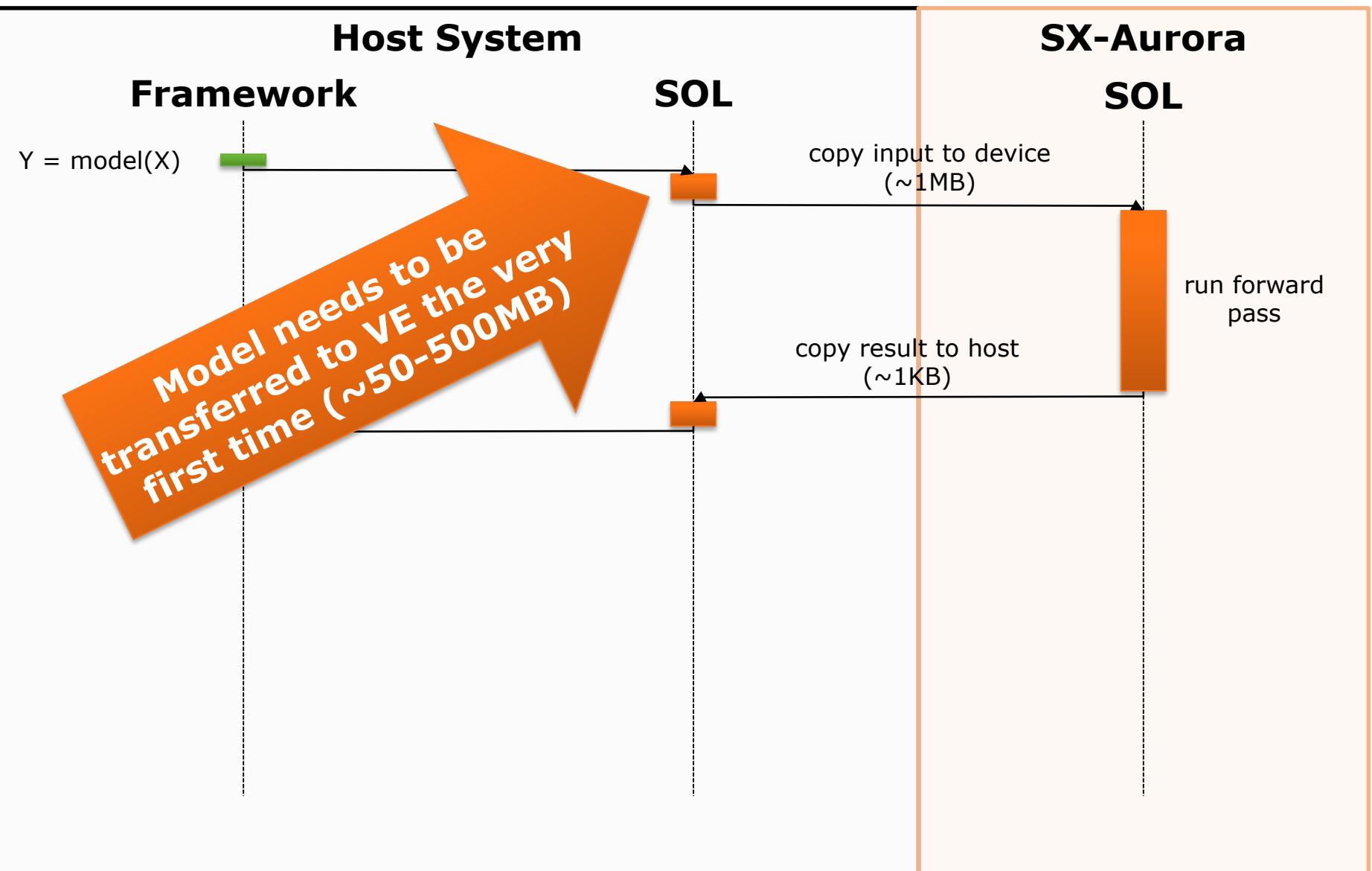
How to integrates SOL into the frameworks?

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

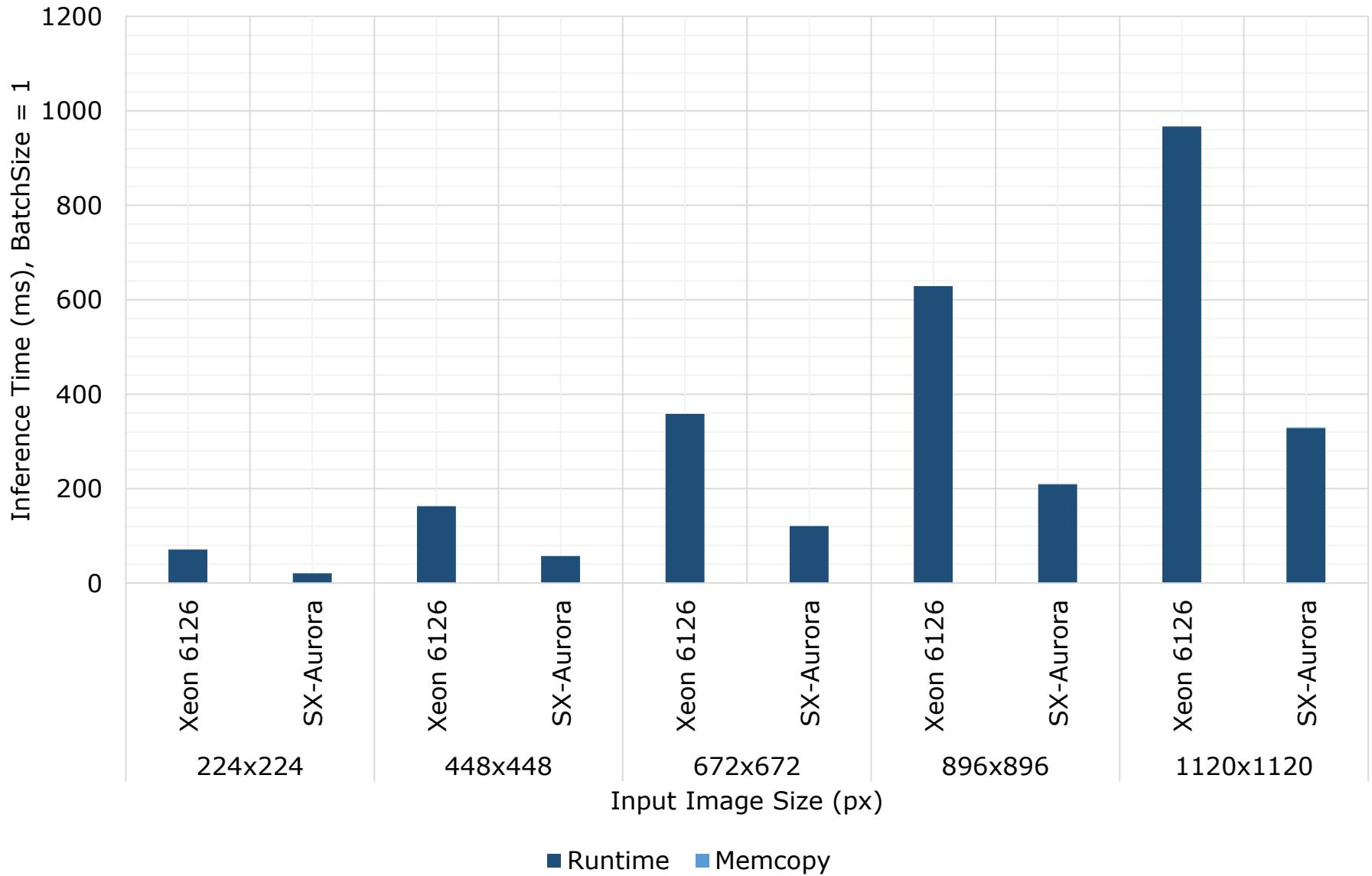
```
class SolLayer(torch.nn.Module):  
    def __init__(self):  
        self.ParamA = ...  
        self.ParamB = ...  
  
    def forward(self, X):  
        return sol.run(X, self.ParamA, self.ParamB)
```



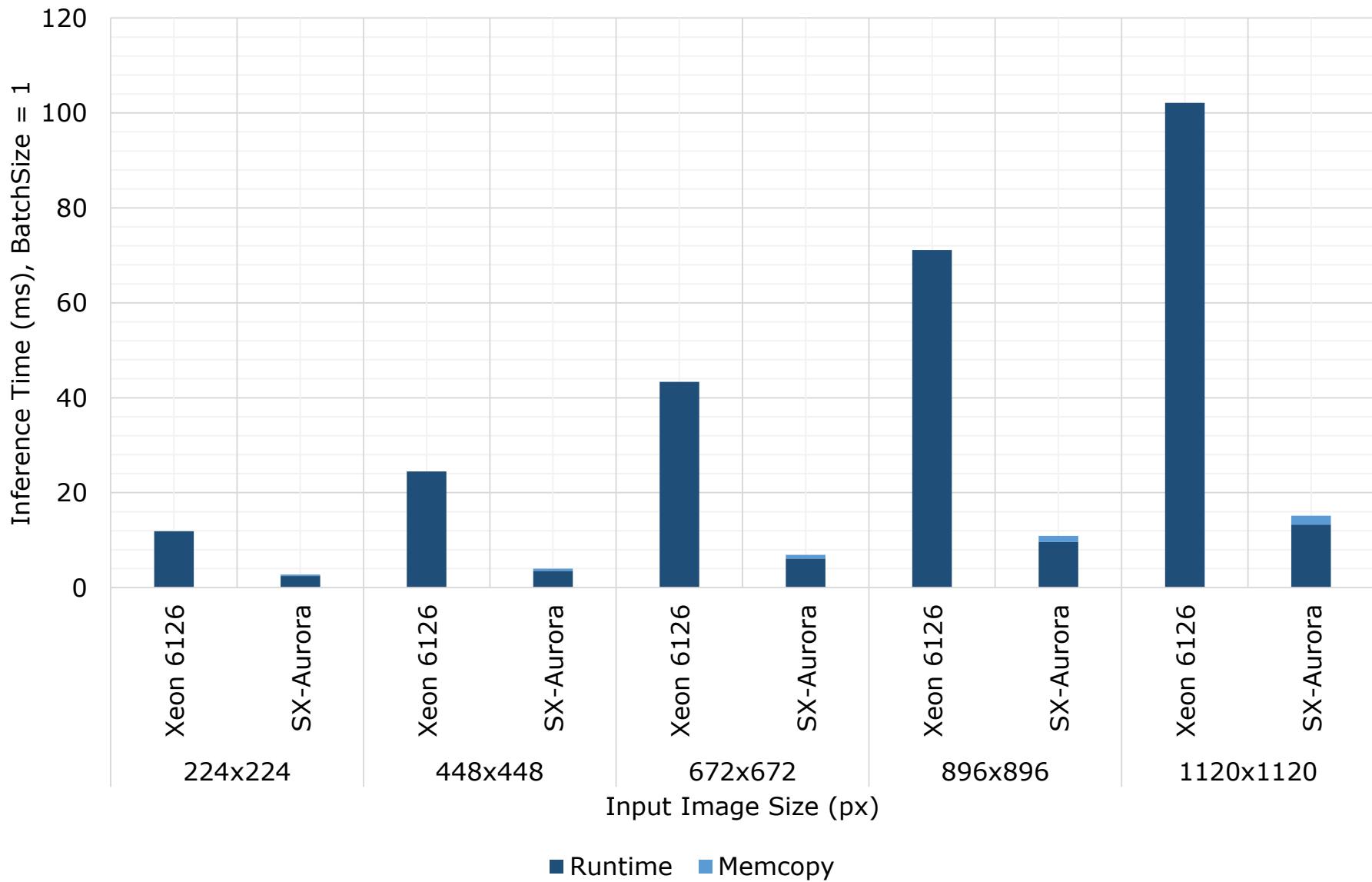
SOL SX-Aurora Inference



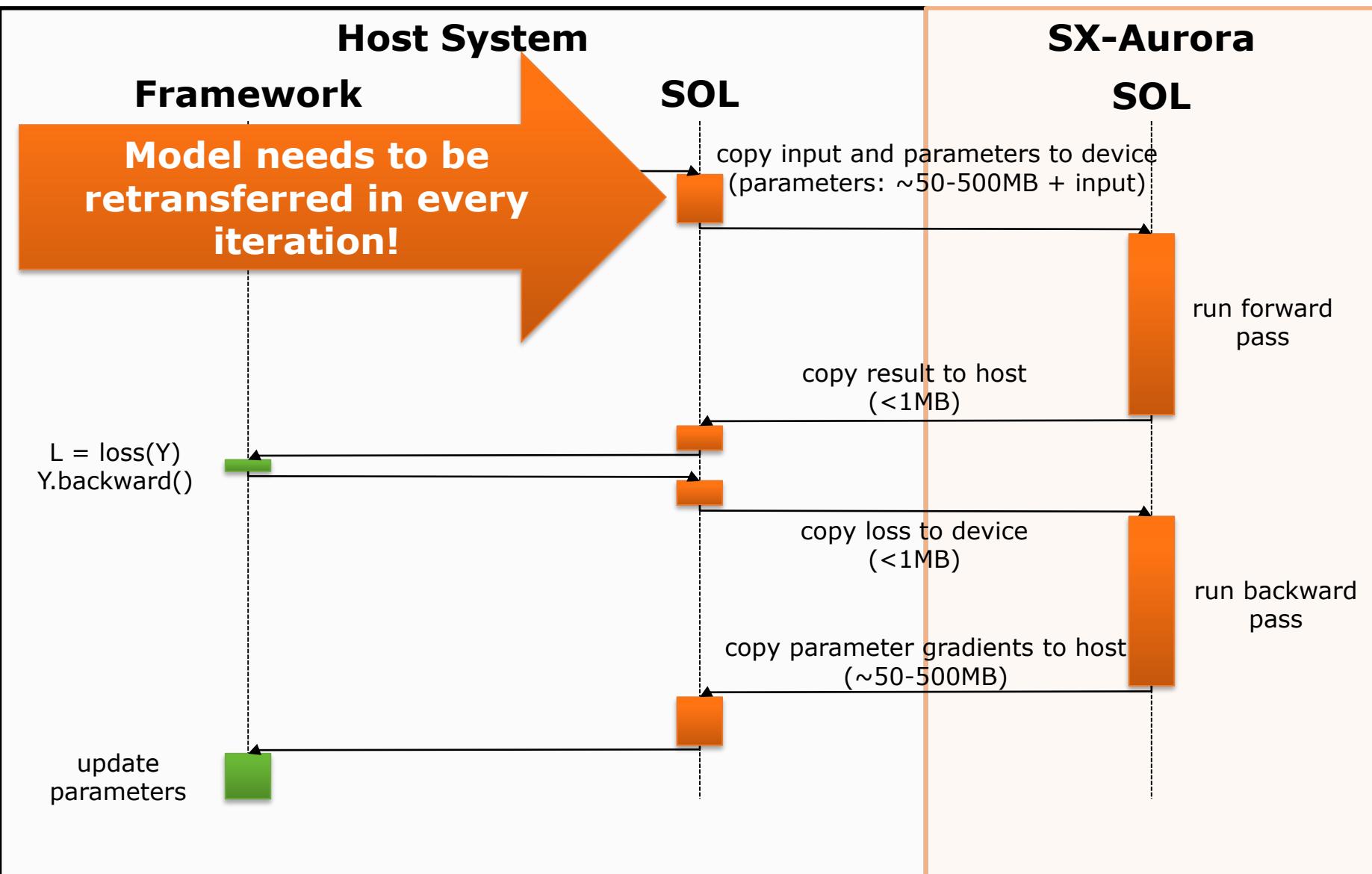
Inference on DenseNet 121



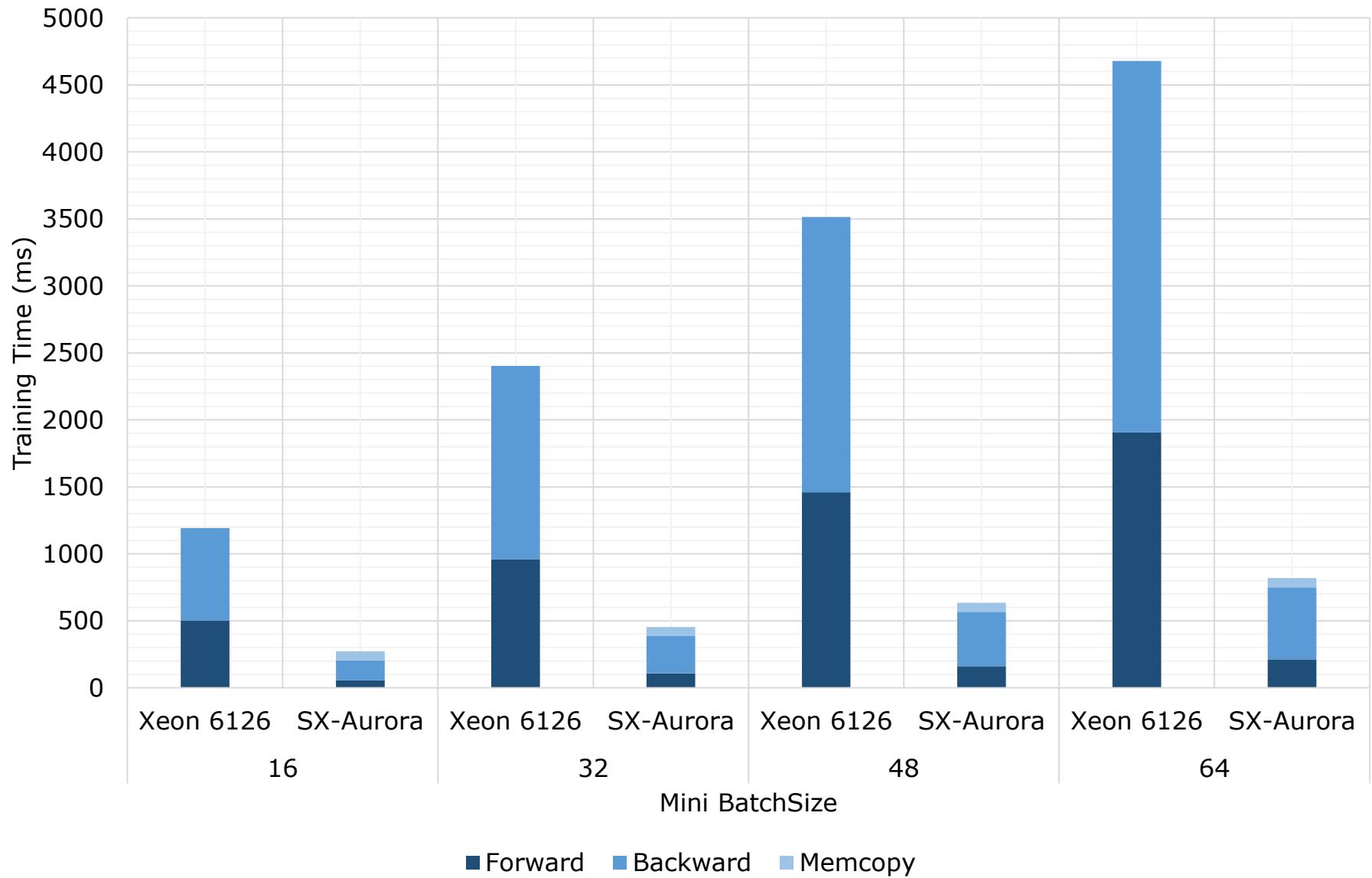
Inference ShuffleNet v2/0.5



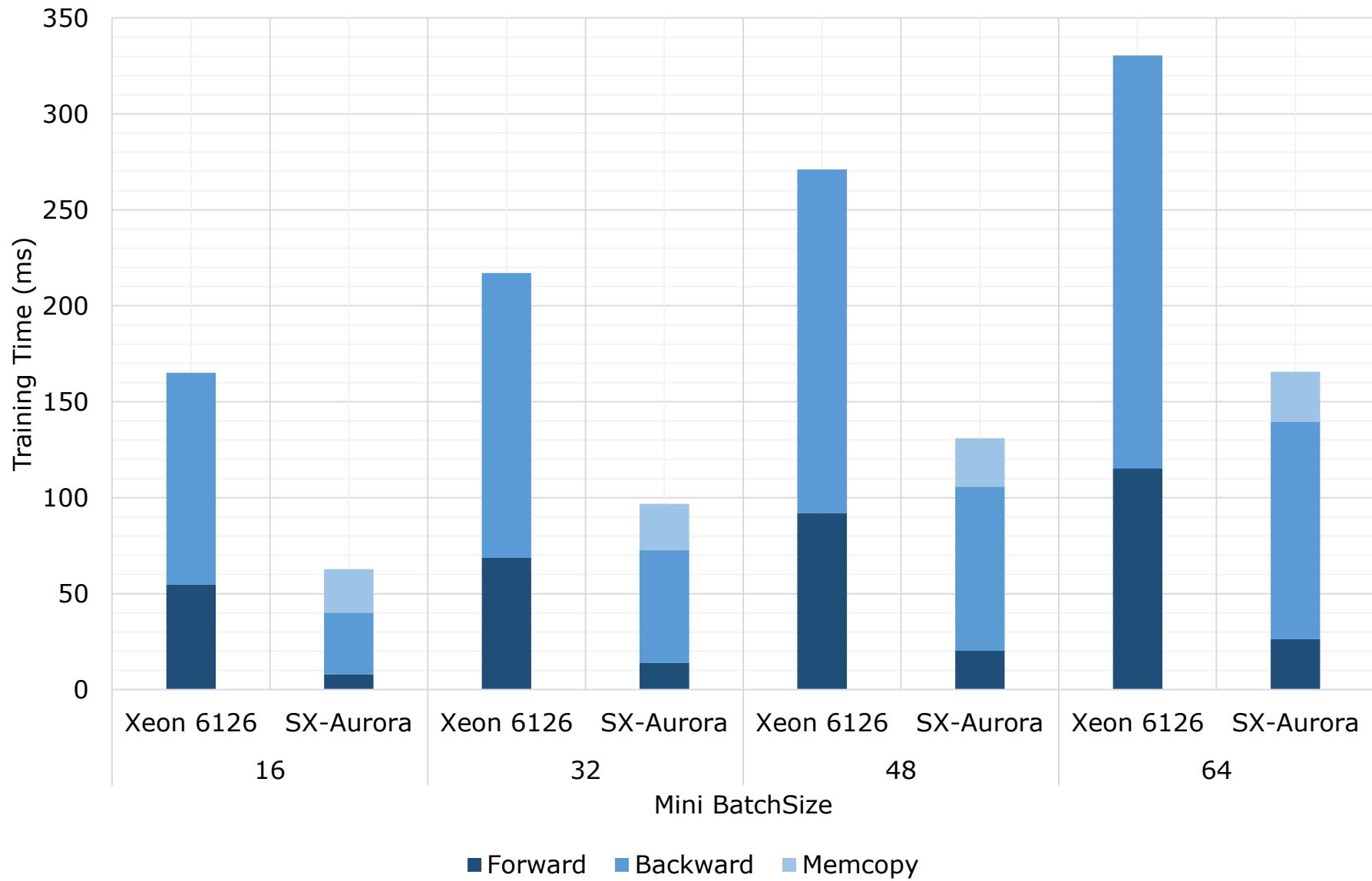
SOL SX-Aurora Training



Training on DenseNet 121



Training on ShuffleNet v2/0.5



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.ve,  
lib_name="MyNetwork", func_name="predict", ...)
```

```
#ifndef __MyNetwork__  
#define __MyNetwork__  
  
#ifdef __cplusplus  
extern "C" {  
#endif  
  
void predict_init(const int deviceIdx);  
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*  
ParamA, const float*  
float* dParamB,  
           /* YOUR  
           */  
           const float*  
           float* dParamA,  
           }  
           }
```

**coming
in 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



SysML Projects ▾ Publication and Talks GitHub Contact Us

SOL

Transparent Neural Network Acceleration

We are HIRING!

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

The SOL project 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 Researcher

NEC Laboratories Europe

nicolas.weber@neclab.eu

www.sol-project.org



The graphic features a green background with a complex network of white nodes connected by lines, resembling a neural network or a molecular structure. The text "NEC HPC platform" and "Deep Learning" is overlaid on the left side. On the right, there is descriptive text about the growth of interest in Deep Learning and its applications in various fields like computer vision and natural language processing.

NEC - HPC Companion 2019/20 NEC HPC platform for Deep Learning

NEC HPC platform for Deep Learning NEC - HPC Companion 2019/20

NEC HPC platform
Deep Learning

The last few years have seen a growing interest in Deep Learning techniques, thanks to the major breakthroughs achieved in the fields of computer vision and natural language processing. Complex problems, such as object detection, text translation and speech recognition, can now be solved with relatively small effort leveraging Deep Learning techniques.

These successes have encouraged researchers to explore many areas of application for such techniques. For example, even large computation tasks generally belonging to the high-performance computing domain, such as fluid mechanics, biology and astrophysics, may benefit from the application of Deep Learning to improve quality of simulations and increase computation performance.

113

Talk at NEC Booth

Thursday November, 21st 11:00am