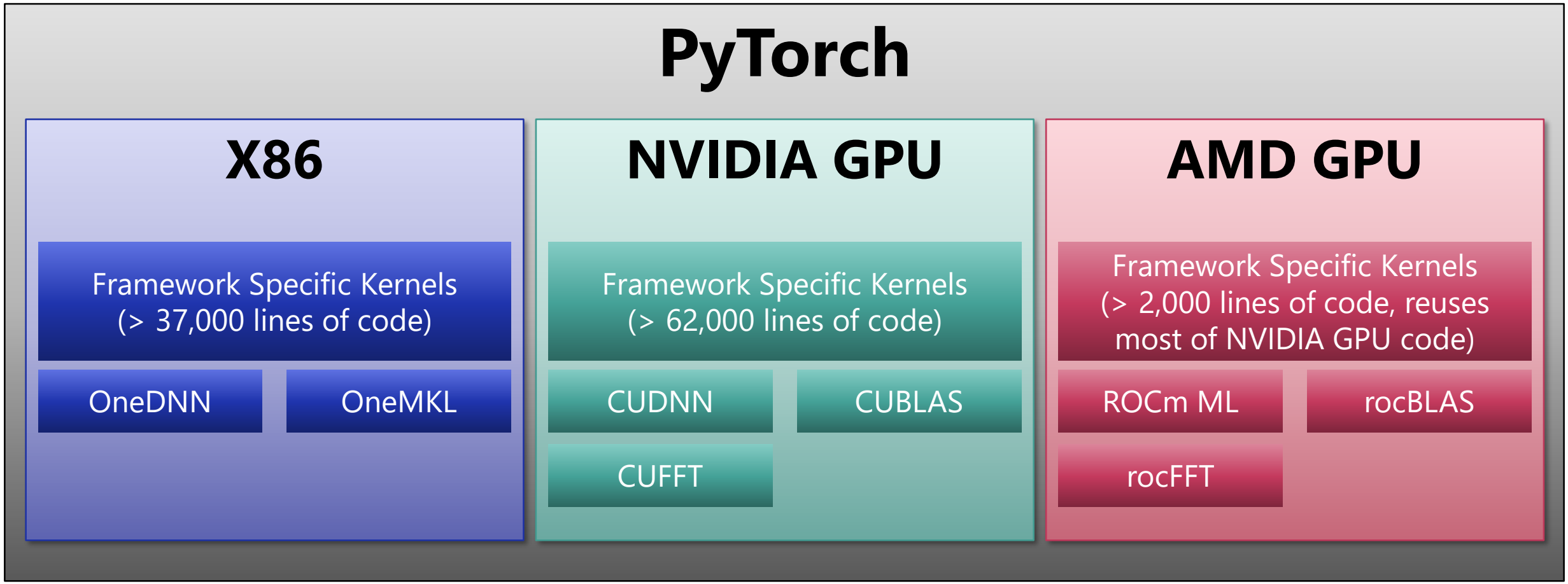
\Orchestrating a brighter world NEC

SOL: Single middleware for optimized multi-architecture AI training and deployment

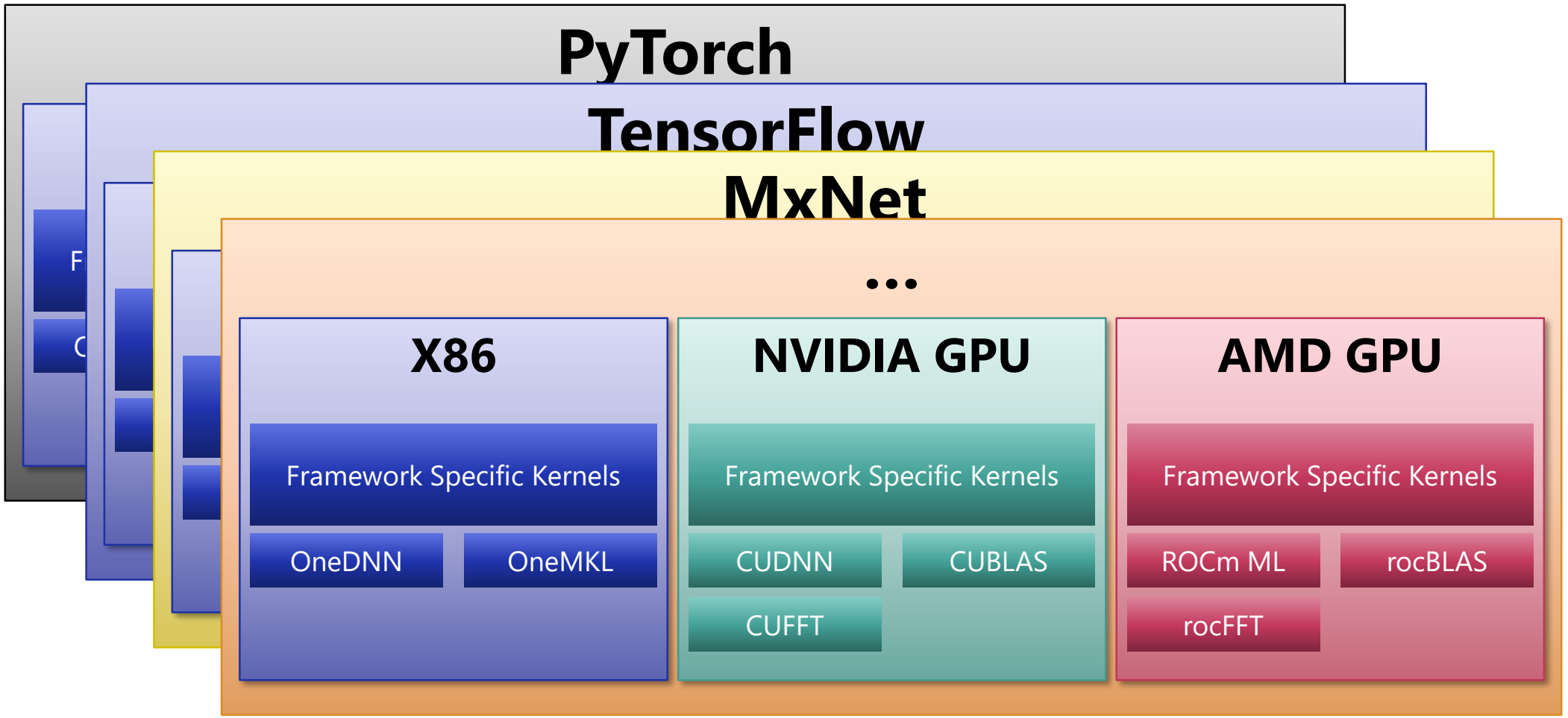
Nicolas Weber <nicolas.weber@neclab.eu>
Intelligent Software Systems
NEC Laboratories Europe

What are the challenges of hardware vendors when integrating their hardware in AI frameworks?

Hardware Support in AI Framework's today

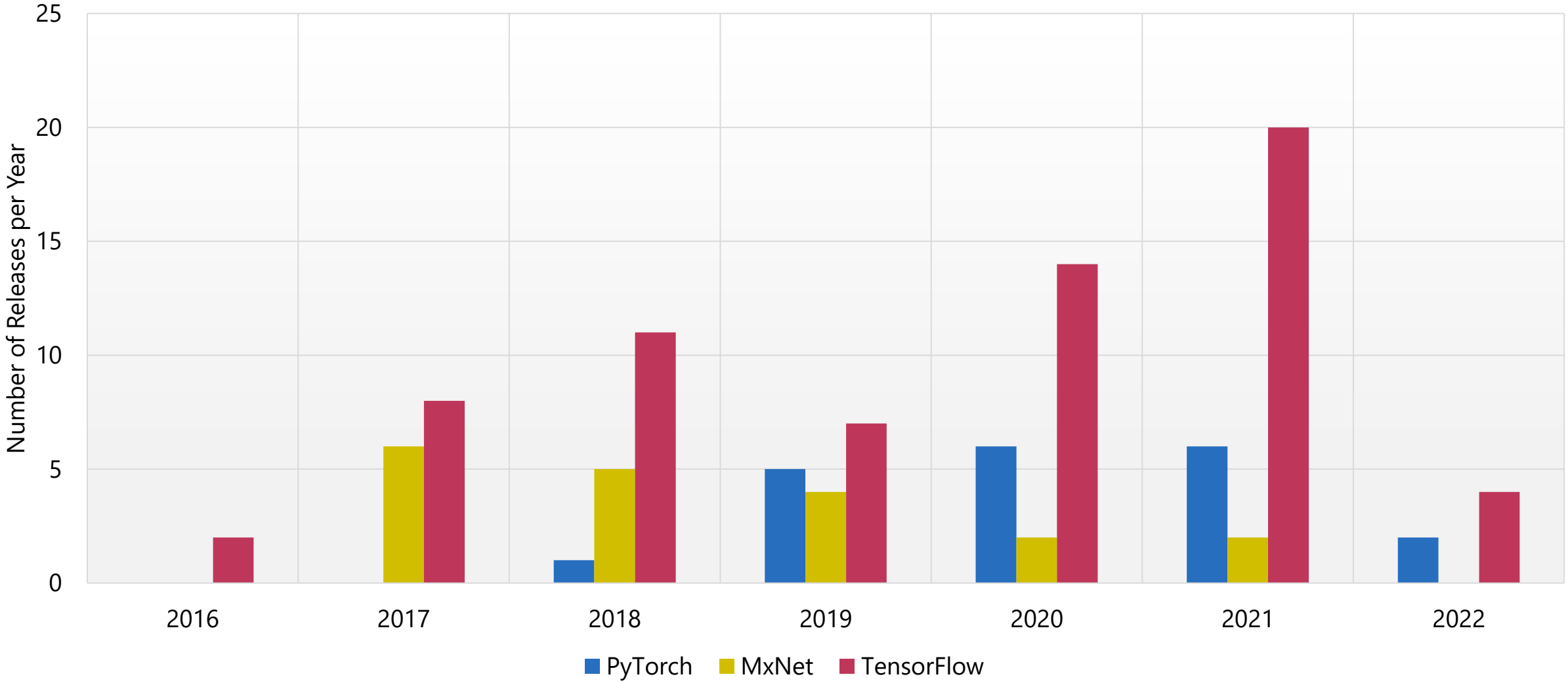


Hardware Support in AI Framework's today



Number of Releases (Major + Minor) per year

(numbers taken from PYPI)



Abstracting AI Frameworks

PyTorch | **TensorFlow** | **ONNX** | **DL4J**

SOL

DFP: Tensor Compiler

X86

Hardware Specific DFP Implementation

OneDNN | OneMKL

Σ 3,700 lines of code

NVIDIA GPU

Hardware Specific DFP Implementation

CUDNN | CUBLAS | CUFFT

Σ 2,100 lines of code

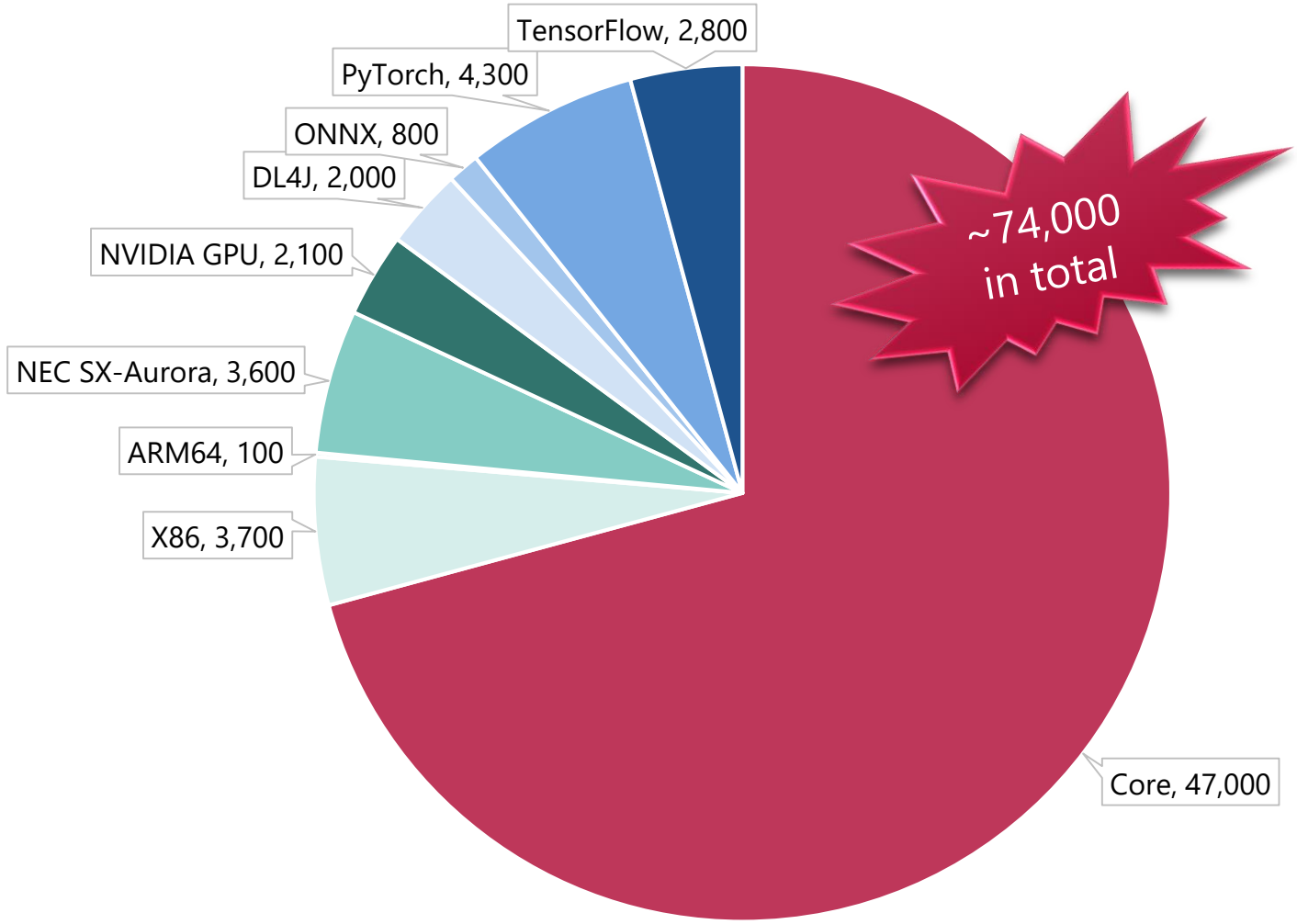
NEC SX-Aurora

Hardware Specific DFP Implementation

VEDNN | NLC

Σ 3,600 lines of code

SOL: Lines of Code per Component



OK that’s nice, but how to use SOL?

Optimizing neural network for Inference

```
# import packages
import torch
import torchvision



# load model and input data
model = torchvision.models.resnet50(pretrained=True)
input = torch.rand(1, 3, 224, 224)



# run inference mode
with torch.no_grad():
    output = model(input)
```

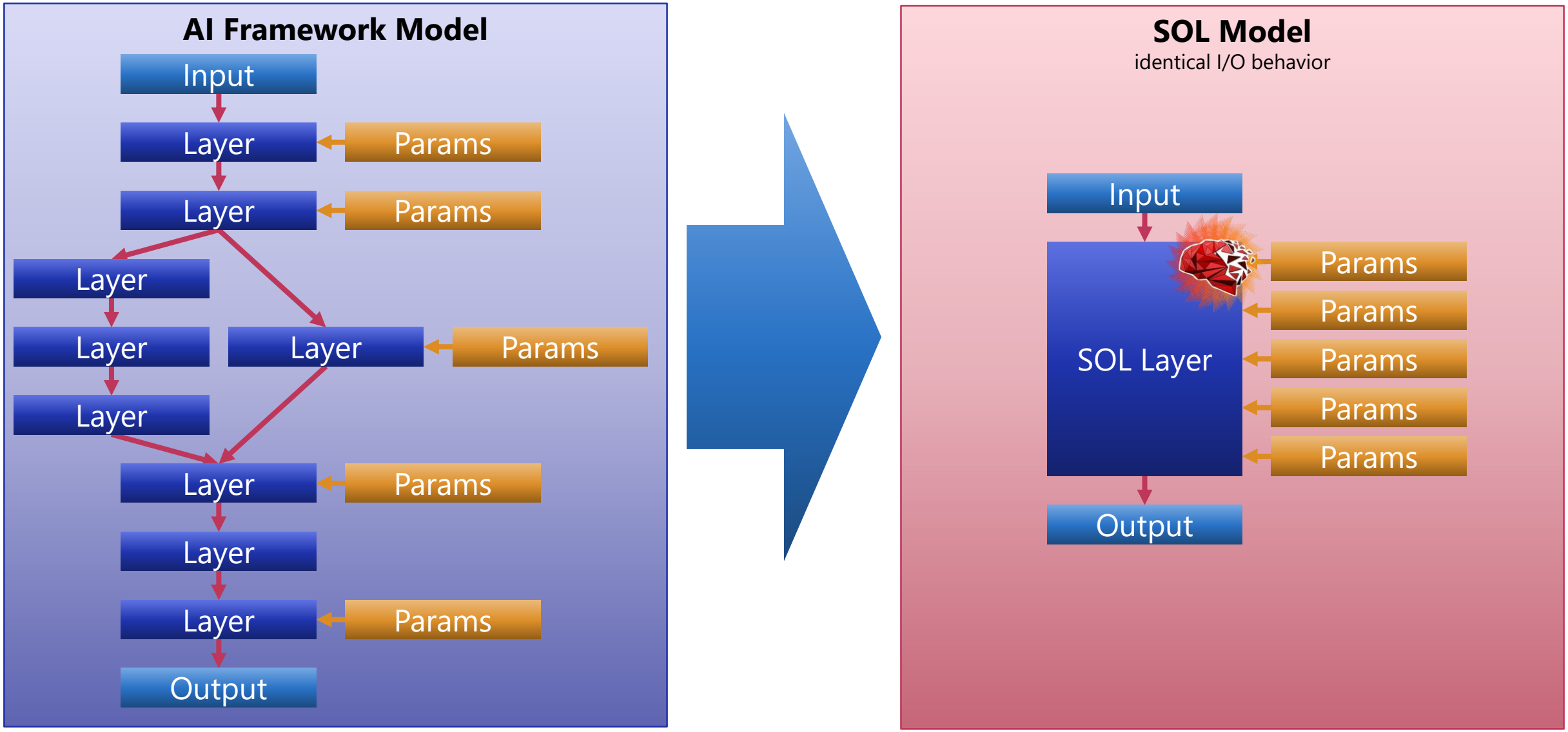
Optimizing neural network for Inference

```
# import packages
import torch
import torchvision
import sol

# load model and input data
model = torchvision.models.resnet50(pretrained=True)
input = torch.rand(1, 3, 224, 224)
model = sol.optimize(model, [input])

# run inference mode
with torch.no_grad():
    output = model(input)
```

sol.optimize(...)



And that’s all that is needed for adding device support to AI frameworks?

Graph vs Eager computations

```
input     = torch.rand(1, 3, 224, 224)
model     = sol.optimize(model, [input])
optimizer = torch.optim.sgd(model.parameters())

for input, target in dataset:
    output = model(input)
    loss = loss_func(output, target)
    loss.backward()
    optimizer.step()
```

Graph computations << executed + optimized by SOL
Eager computations << executed by AI framework

Eager Computation integration

- **VEDA-[PyTorch, TensorFlow]**
  - Separated from SOL since v0.5.0
  - https://github.com/SX-Aurora/veda-[pytorch, tensorflow]
  - Open Source License: BSD-3
  - Installation via: `pip3 install veda-[pytorch, tensorflow]~=[FRAMEWORK-VERSION]`
    (requires explicit linking against specific framework version)
  - Contains minimal subset of eager computations needed to run inference + training workloads in PyTorch and TensorFlow.

- **VEDA-PyTorch (1.10.2 and 1.11.0 supported)**
  - Required code: `import veda.pytorch`
  - Enables "VE" device and `tensor.ve(deviceIdx)`, `torch.ve.synchronize()`, etc.

- **VEDA-Tensorflow (2.6.3, 2.7.1 and 2.8.0 supported)**
  - Automatically loaded when importing TensorFlow
  - Just use: `with tf.device("/VE:0")`

Transparent Offloading

- Enables to execute workloads on accelerators in frameworks that don't allow to add new native devices, such as Numpy
  (more details: https://arxiv.org/abs/2003.10688)

```
import sol
import numpy as np
model = sol.optimize("my_onnx_model.onnx") # returns numpy a function
input = np.random.rand(1, 3, 224, 224)
sol.device.set("ve", 0)                    # sets VE#0 as executor
output = model(input)                      # computes on VE#0
```

Which other benefits does SOL have for me as a user?

AI Deployment: Runtime Engines

- **Runtime Engines**
  - ONNXruntime
  - LibTorch (PyTorch)
  - TensorFlow Lite
  - …
- **Pros**
  - Easy to use
  - Rapid deployment
- **Cons**
  - No compiler based optimizations
  - Large storage footprint (not including your AI model!)
    - LibTorch 1.11.0 with CUDA: **1.6GB**!
    - onnxruntime-gpu: 110MB

AI Deployment: AI Compilers

- **AI Compilers**
  - TVM (multi-vendor)
  - OpenVINO (all Intel Hardware)
  - TensorRT (NVIDIA)
  - Arhat (Fragata-AI)
  - …
- **Pros**
  - Optimized performance
  - Small storage footprint (only required functionality is shipped)
- **Cons**
  - usually vendor-locked (except TVM)
  - no guarantee that all functionality of your AI framework can be used
    (i.e. compilers usually use ONNX as input, but ONNX does not support complex dtypes, FFT, …)

AI Deployment: SOL

Pros

- Uses SOL optimizing compiler engine
- Supports ALL functionality of SOL (including FFT, complex dtypes, …)
- Multi-vendor
- Fully customizable runtime API
  (i.e. implement your own memory allocator)
- Allows static and shared linking

Cons

- still in experimental state
- not yet available for v0.5.0 release candidates, but will be released this summer!

Cross-Framework execution

- Enables to run foreign models, i.e., PyTorch model in TensorFlow!

```
import torch
from torchvision import models as tv
import sol
import numpy as np
import tensorflow as tf

with tf.device("VE:0"):
        input  = np.random.rand(1, 3, 224, 224)
        model  = sol.optimize(tv.resnet50(), [torch.from_numpy(input)], framework="keras")
        output = model.predict(input)
```

- **Available executors**
  - "pytorch" (torch.nn.Module)
  - "tensorflow" (tf.Module)
  - "keras" (tf.keras.model.Model)
  - "numpy" (Python function expecting inputs as np.ndarray)

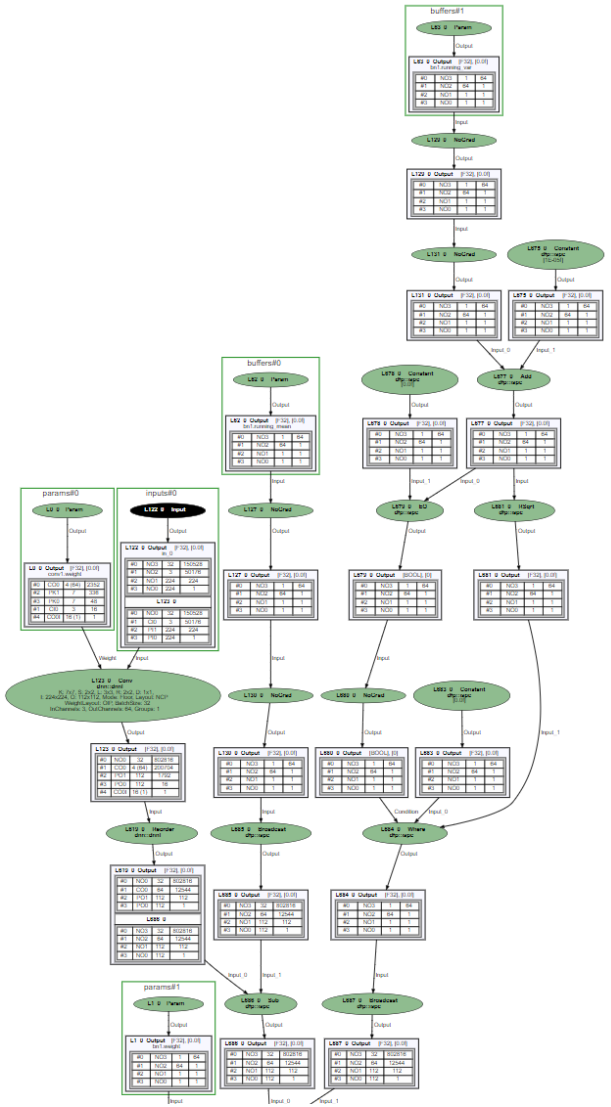
Additional Information

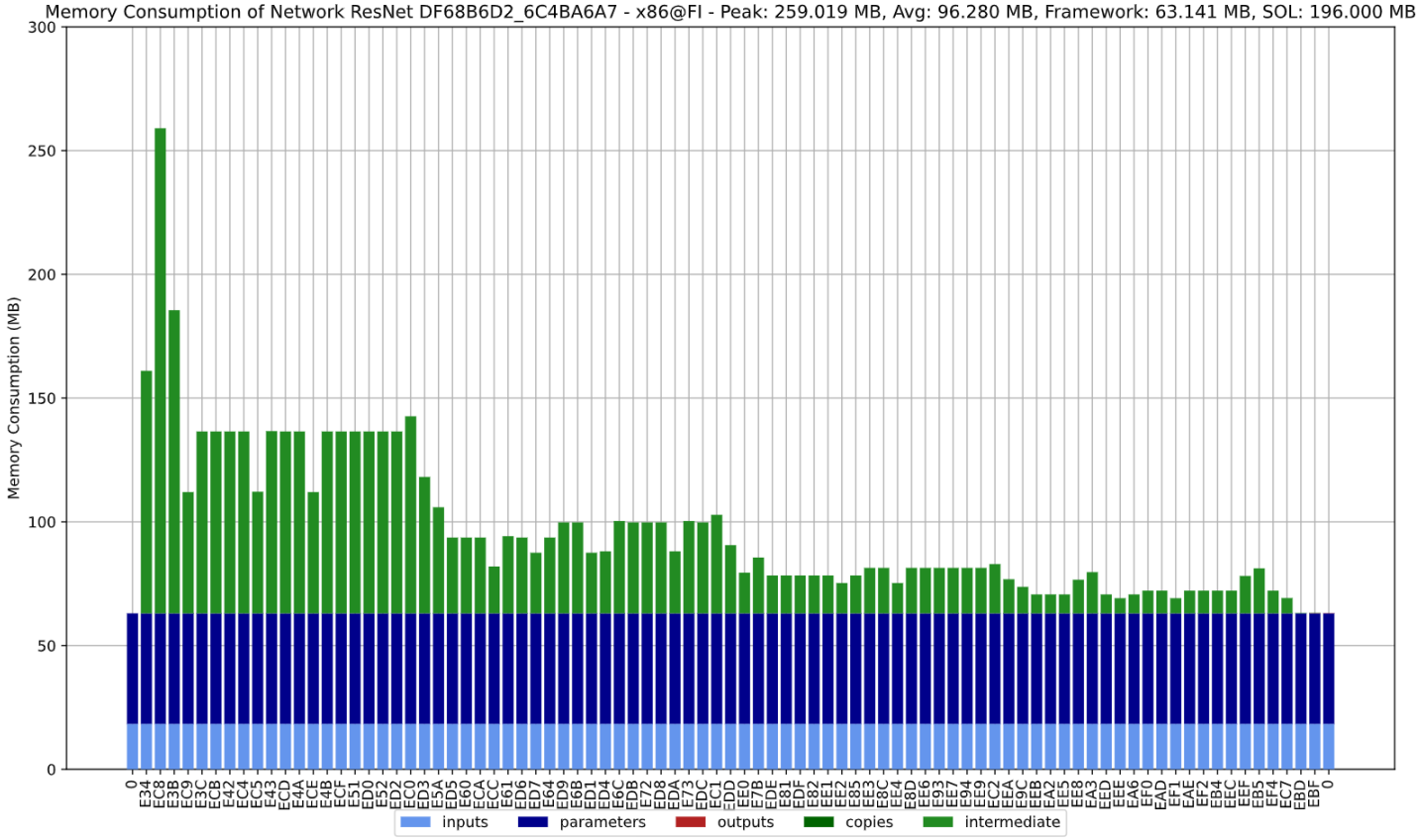
- Model Analysis (Input/Output Shapes, Model Parameters)
- Memory Consumption Estimation

```
Analyzing network ResNet (0xDF68B6D2)
Inputs:
    in_0: Tensor(dtype=[F32], shape=[32, 3, 224, 224])
Outputs:
    Tensor(dtype=[F32], shape=[32, 1000])
Model Parameters: 44.63MB
100% [##########################################################################]

Compiling network DF68B6D2_6C4BA6A7 for x86
100% [##########################################################################]
Estimated Peak Memory Consumption:
    Inference: ~260MB
    Training:  ~970MB
```

Additional Information: Computation Graph Visualization





That’s amazing! How can I get my hands on it?

How to get started?

- SOL is still in closed beta, BUT you can participate by requesting access through your NEC sales representative, Erich Focht or me
- Install SOL installer: `pip3 install --pre nec-sol`
- Use command-line installer

```
## NEC-SOL Package Manager v0.5.0rc1

Please choose an action:
> install/modify modules
  download modules
  list installed modules
  uninstall all modules
  options
  exit installer
```

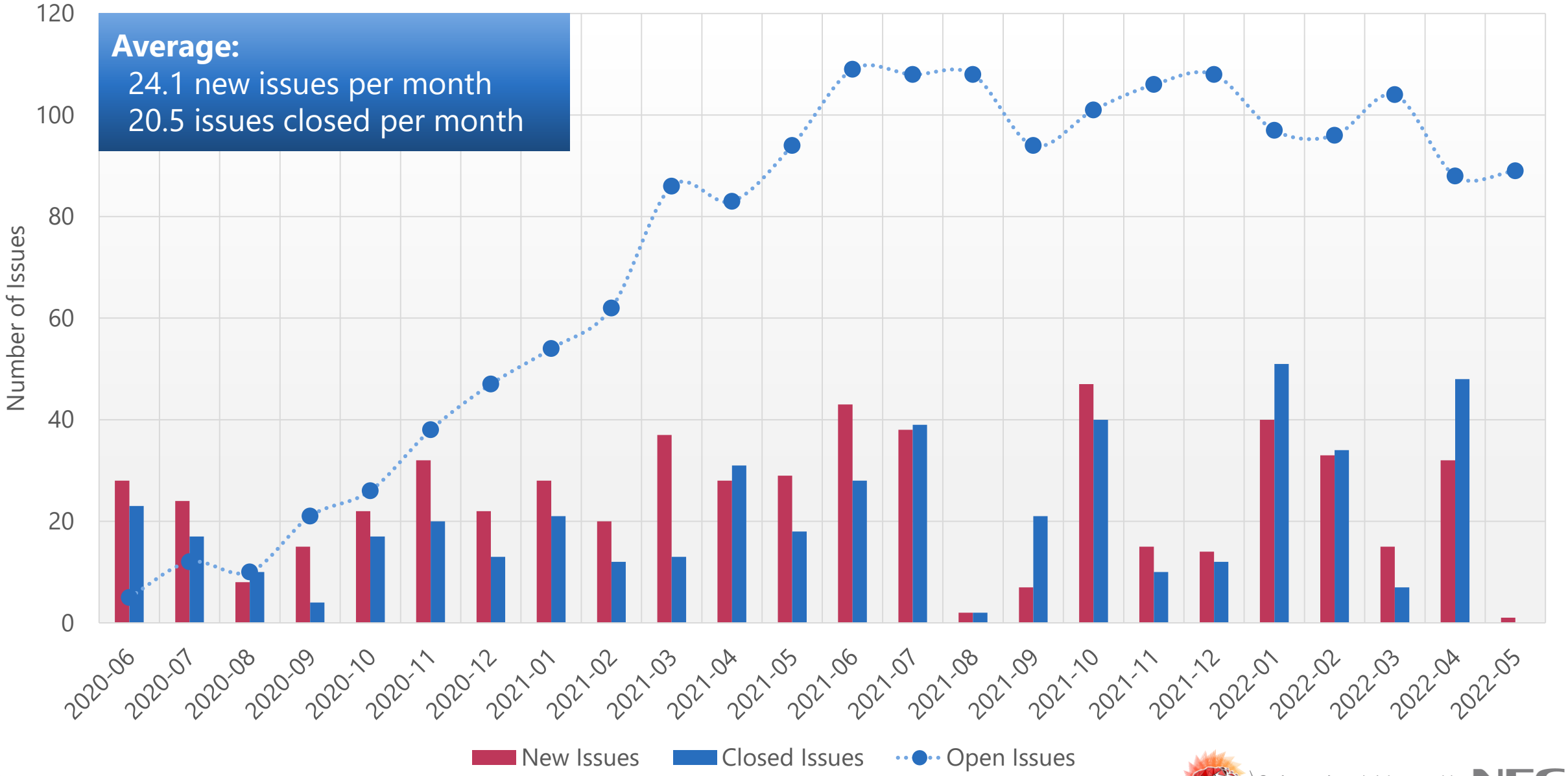
- More information in our documentation: https://sol.neclab.eu/docs

Upcoming SOL v0.5 Diadem release

- Since v0.5 we are using a "rolling releases"-model with release candidates every 2-4 months.

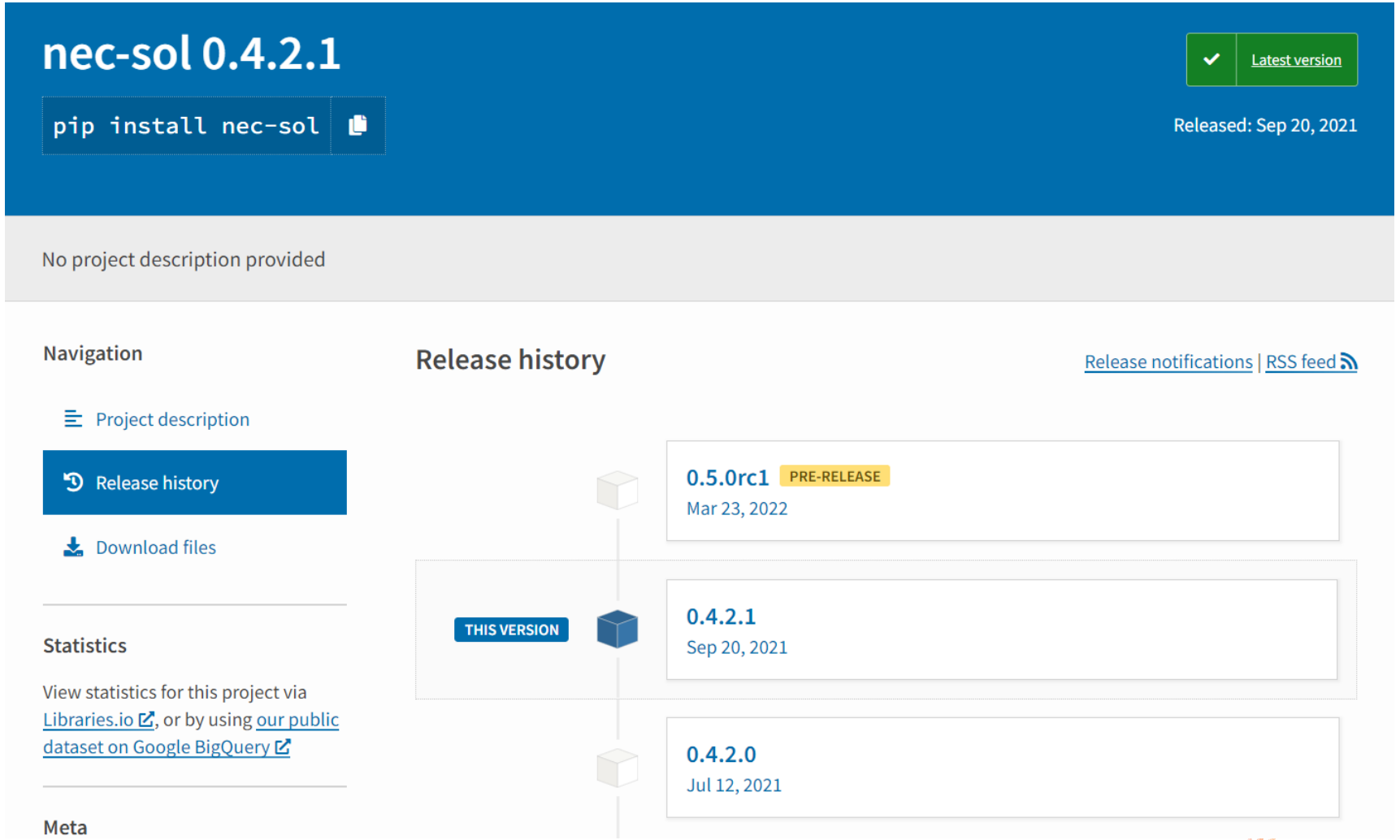
Feature	v0.5.0rc1 (March'22)	v0.5.0rc2 (May'22)	v0.5.0rc3 (Summer'22)
FFT	✓ X86 ✓ VE		
	#N/A		✓ NVIDIA
ONNX	#N/A	✓ upgraded to API-level 13 ✓ added more layers	
New Deployment System	#N/A		✓ static lib ✓ shared lib ✓ customizable API
Cross-Framework Execution	✓ PyTorch ✓ TensorFlow (Module + Keras)		
	#N/A	✓ Numpy	
New Dynamic Dimensions System	✓ see https://sol.neclab.eu/docs/v0.5.0/frameworks.html#variable-dimensions		

SOL Issue Tracker (https://gitlab.neclab.eu)



How to get notified about new versions?

- Subscribe to: https://pypi.org/rss/project/nec-sol/releases.xml



Thank you!

Nicolas Weber

Senior Research Engineer
Intelligent Software Systems Group
NEC Laboratories Europe
nicolas.weber@neclab.eu
sol.neclab.eu

We are looking for **new colleagues**, **visiting researchers** and **students**! Contact me if you are interested in working with us on HPC, Compiler, Scientific Computing, and other related topics!