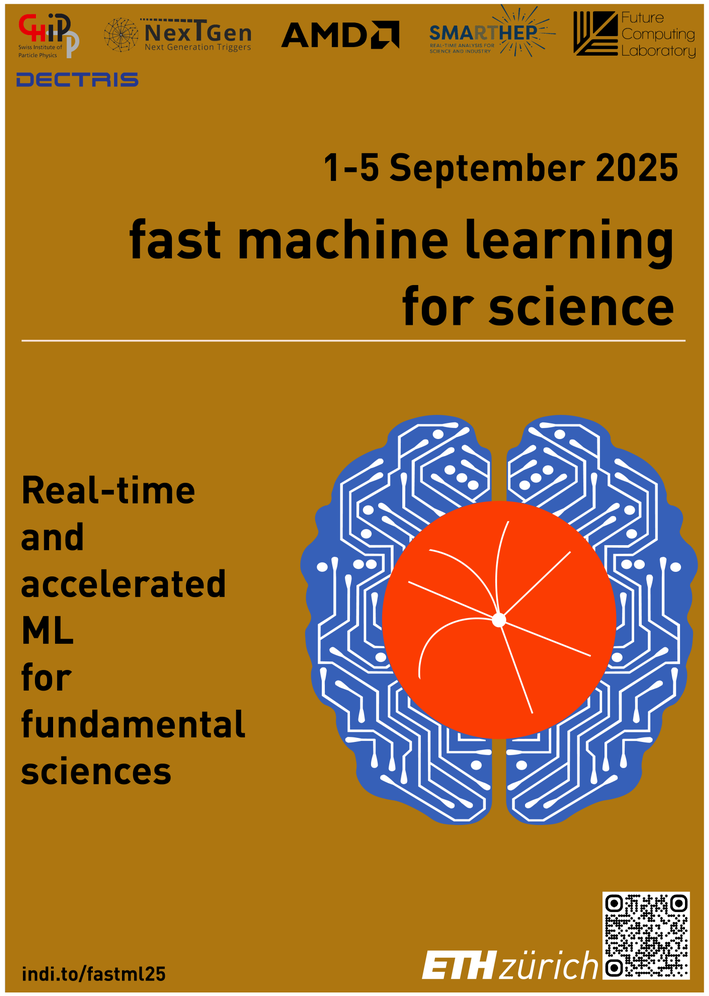
Fast ML for Science

Nhan Tran, Fermilab
AI4EIC 2025

https://indico.cern.ch/e/fastml2025



Fast ML for Science at its core

"Scientific discoveries come from groundbreaking ideas and the capability to validate those ideas by testing nature at new scales - finer and more precise temporal and spatial resolution. This is leading to an explosion of data that must be interpreted, and ML is proving a powerful approach. The more efficiently we can test our hypotheses, the faster we can achieve discovery. To fully unleash the power of ML and accelerate discoveries, it is necessary to embed it into our scientific process, into our instruments and detectors."

Applications and Techniques for Fast Machine Learning in Science
https://doi.org/10.3389/fdata.2022.787421

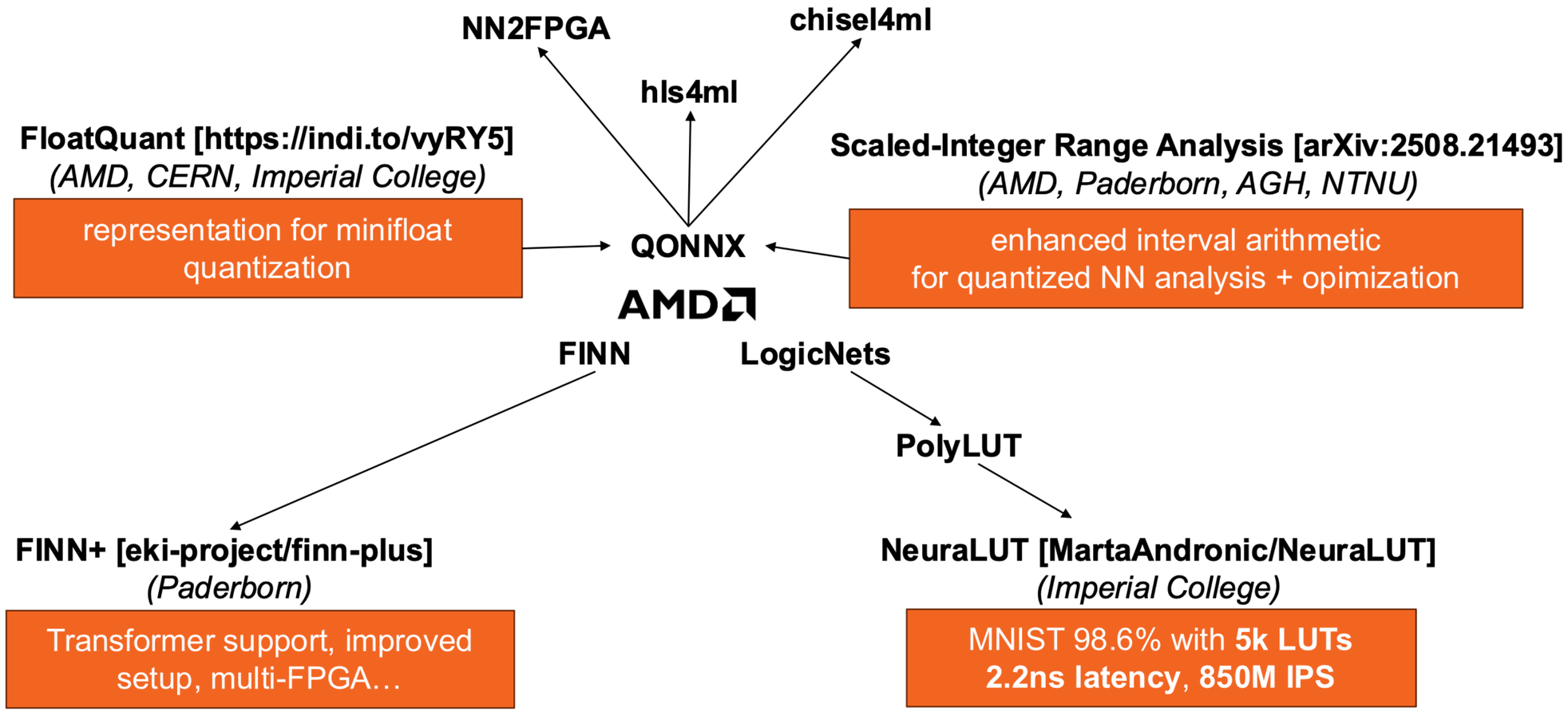
Reflections - a retrospective

- Essence of community:
  - Sharing open-source tools, techniques, and enabling collaboration encouraging cross-disciplinary research
  - User- and application-driven focus to enable science and technology research

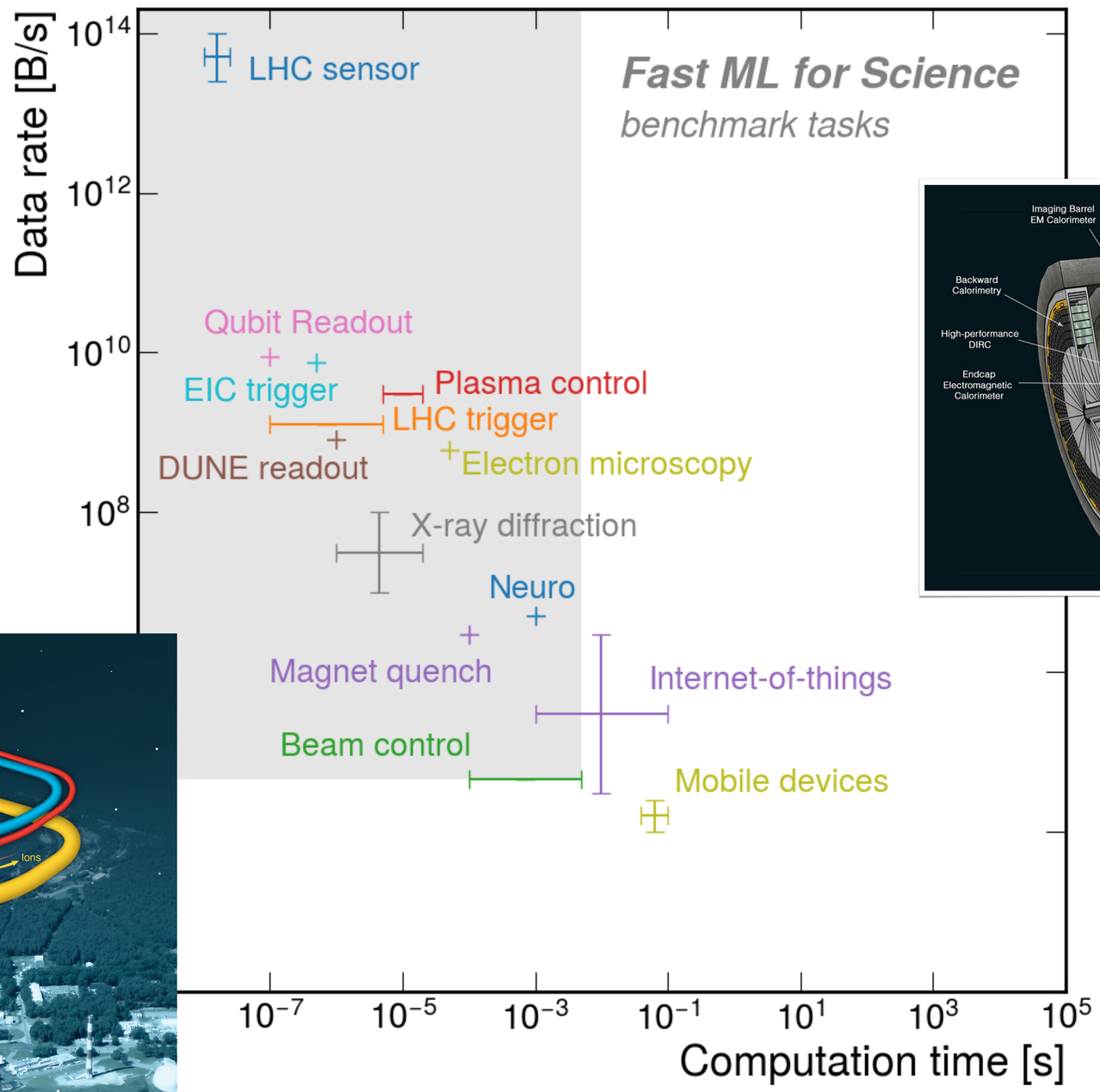
- It can be jarring, but it can build new and powerful synapses!

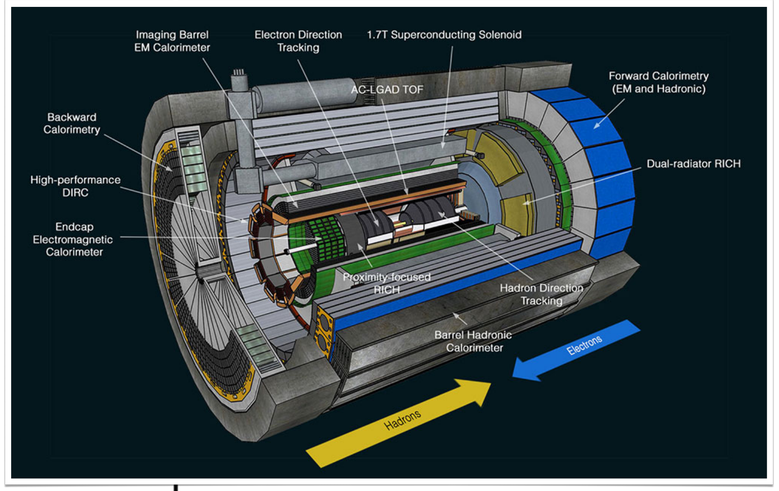
[Public]

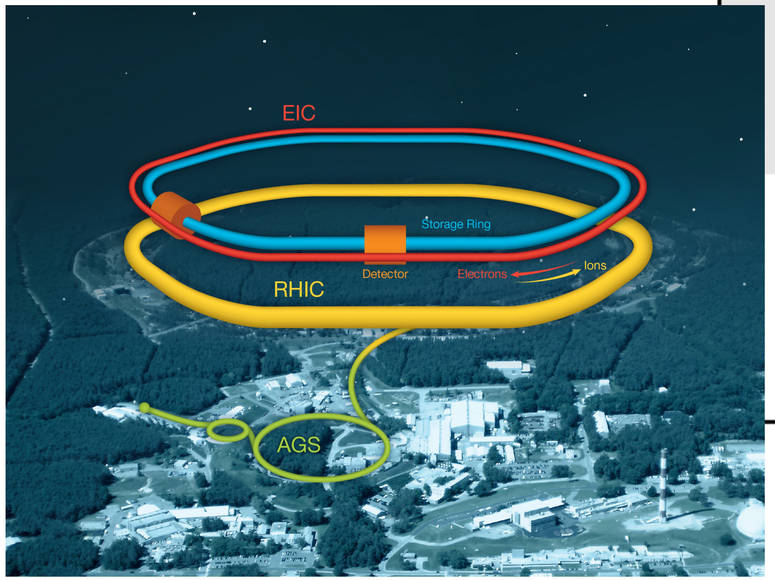
The Power of the Open Ecosystem: Recent Examples



AMD
together we advance_


Fast ML for Science
benchmark tasks
Data rate [B/s]
Computation time [s]
LHC sensor
Qubit Readout
EIC trigger
Plasma control
LHC trigger
DUNE readout
Electron microscopy
X-ray diffraction
Neuro
Magnet quench
Internet-of-things
Beam control
Mobile devices


Imaging Barrel EM Calorimeter
Electron Direction Tracking
1.7T Superconducting Solenoid
AC-LGAD TOF
Forward Calorimetry (EM and Hadronic)
Backward Calorimetry
Dual-radiator RICH
High-performance DIRC
Endcap Electromagnetic Calorimeter
Proximity-focused RICH
Hadron Direction Tracking
Barrel Hadronic Calorimeter
Electrons
Hadrons


EIC
Storage Ring
Detector
Electrons
Ions
RHIC
AGS

Edge of Tomorrow today:

AI at the speed of nature

AI-powered embedded devices that deliver ultra-efficient inference in extreme environments for real-time intelligent sensing and control.

The dream: self-driving experiments that are ultra-fast, precise, and robust

What are the limitation of AI / DL ~~(in robotics)~~?

- One task at a time
  - Need a much more multi-modal approach (not just images and text)
    - Different sampling rates
    - Different dimensionalities
    - Different reliability
    - Different (changing?) geometrical arrangement
  - Orchestrating different tasks is a challenge
- Data hungry
  - Physical world is fundamentally complex and dynamic
- High latency
  - Control loop might need <10ms latency and confidence
- Reliability: accuracy and robustness

Image credit: A. Thea

Fast and slow

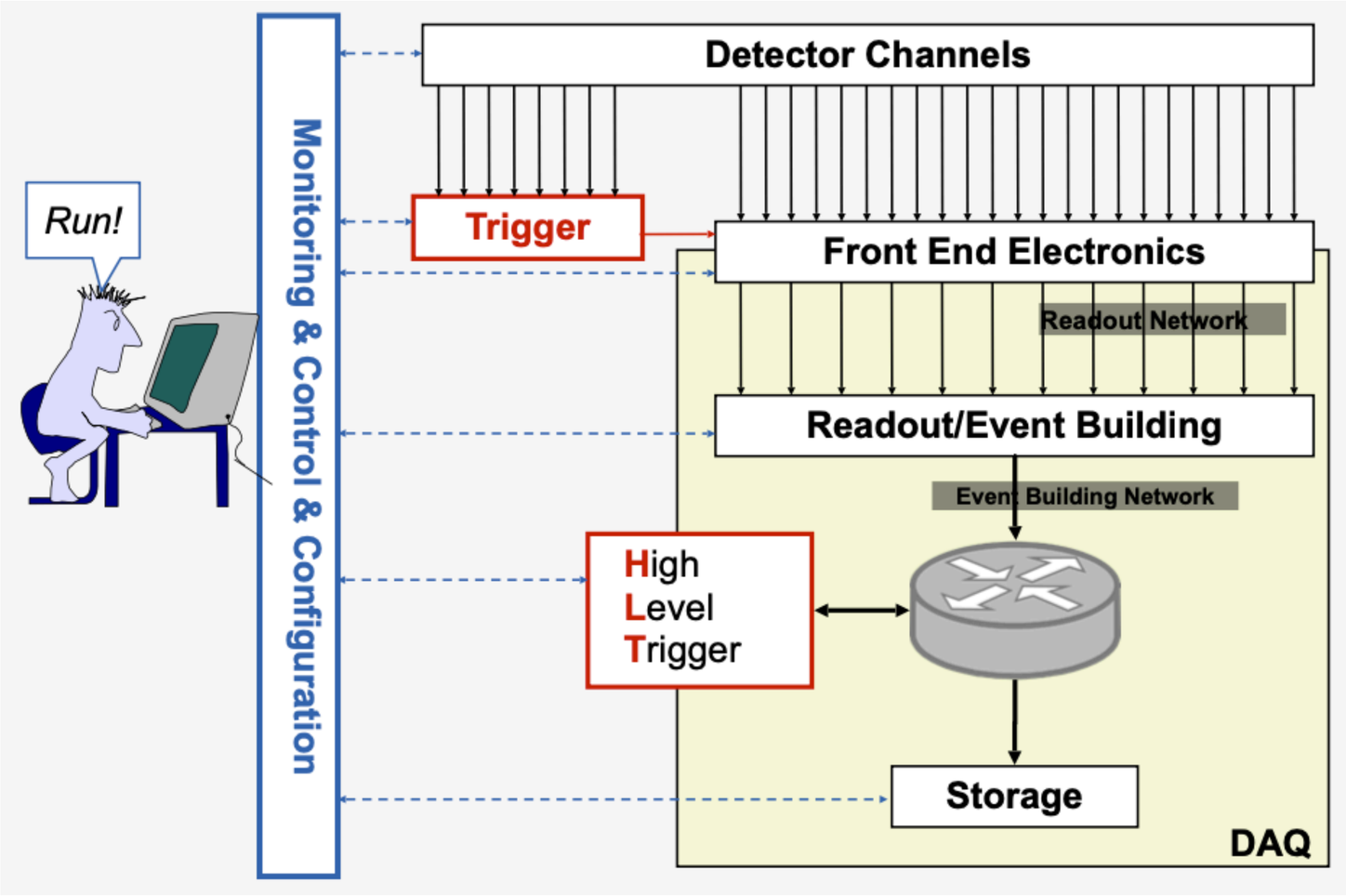


Image credit: A. Thea

Fast and slow

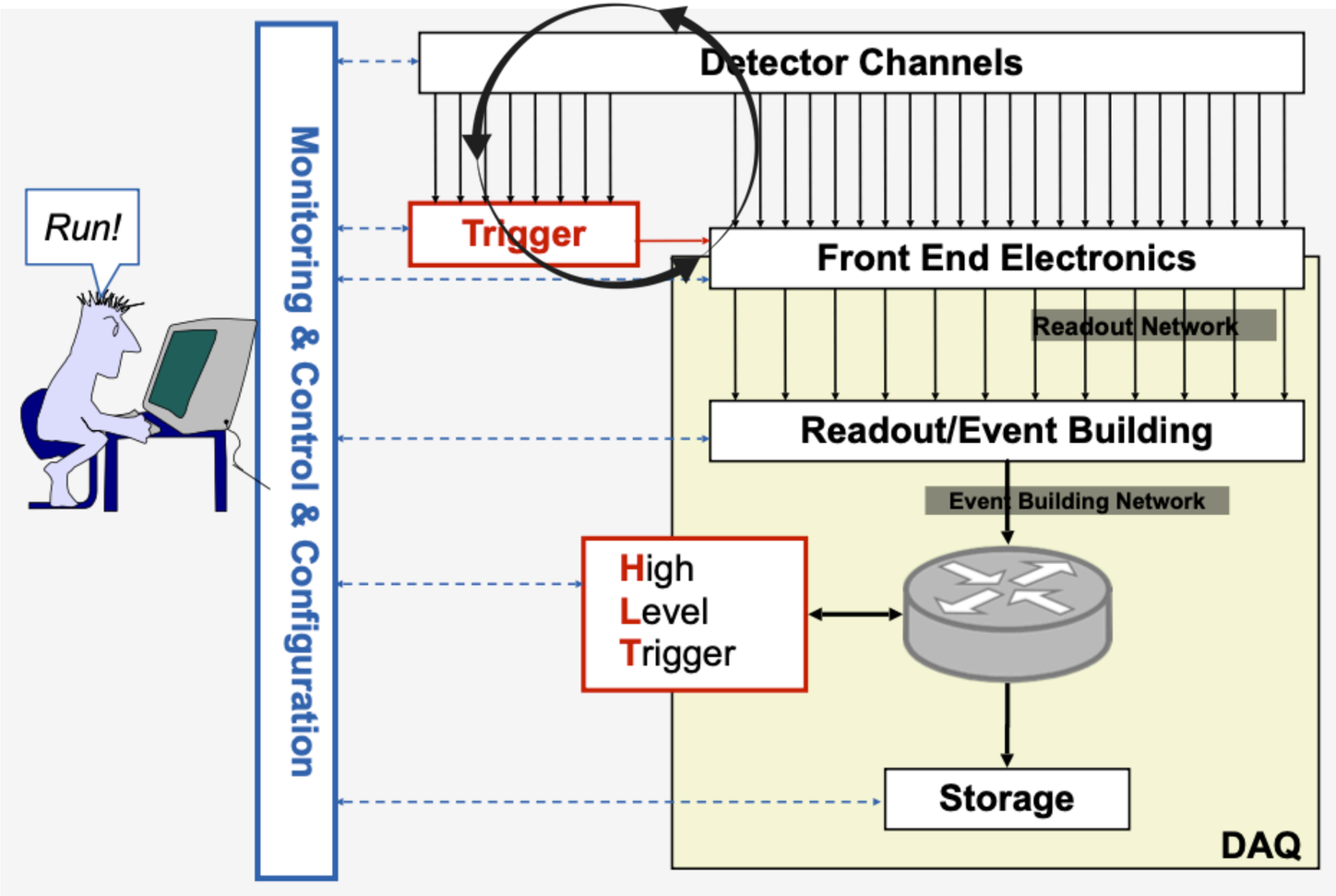


Image credit: A. Thea

Fast and slow

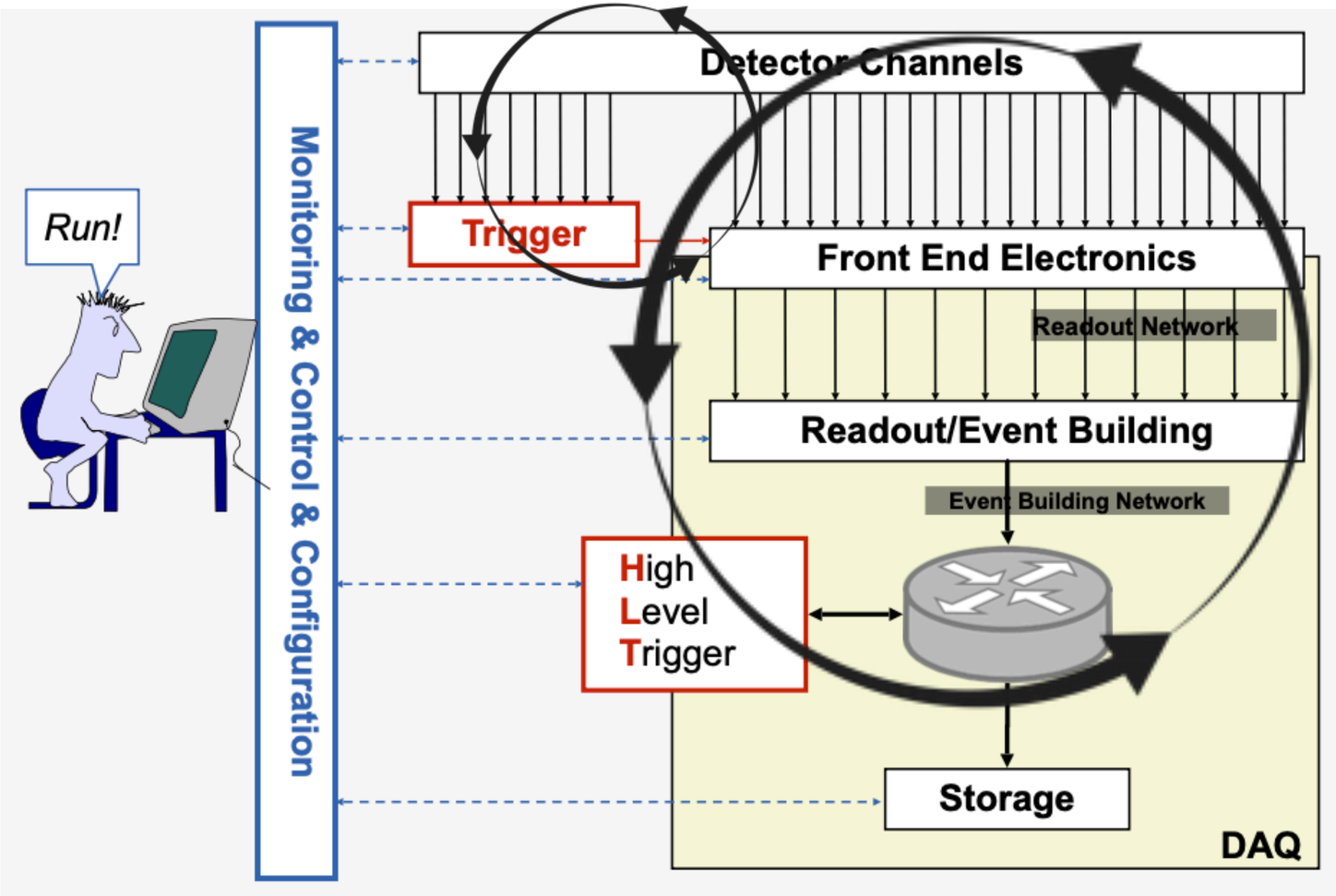


Image credit: A. Thea

Fast and slow

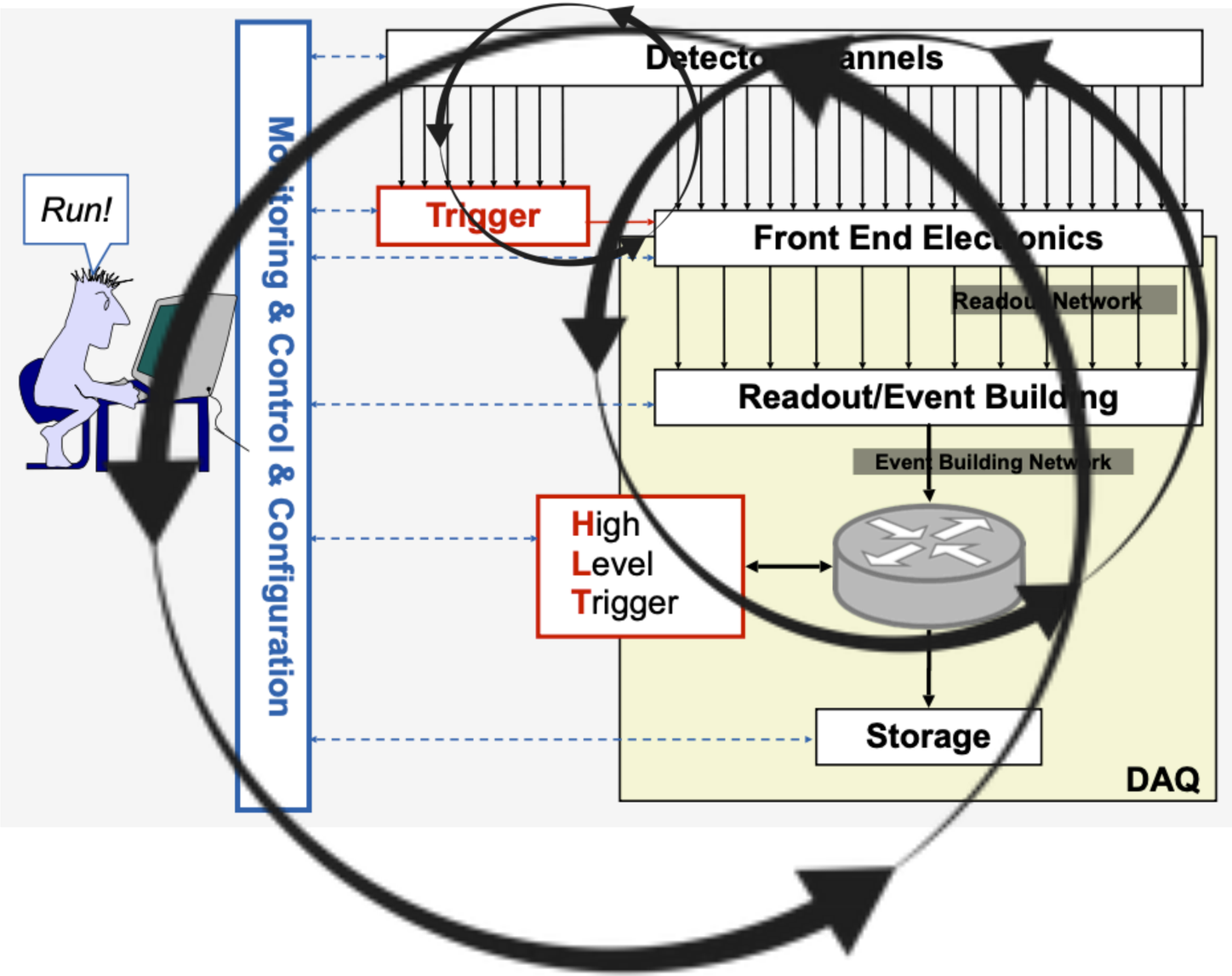
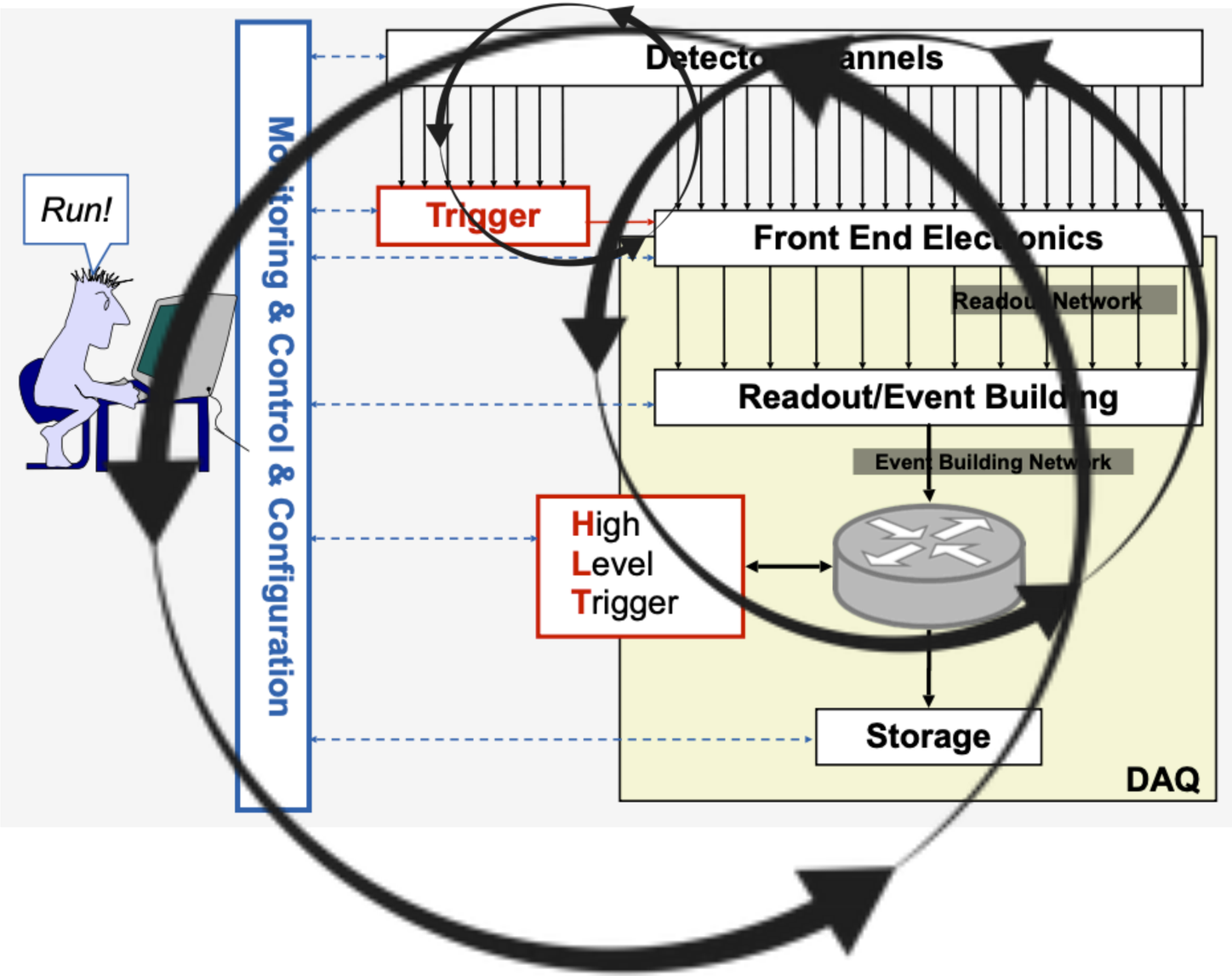
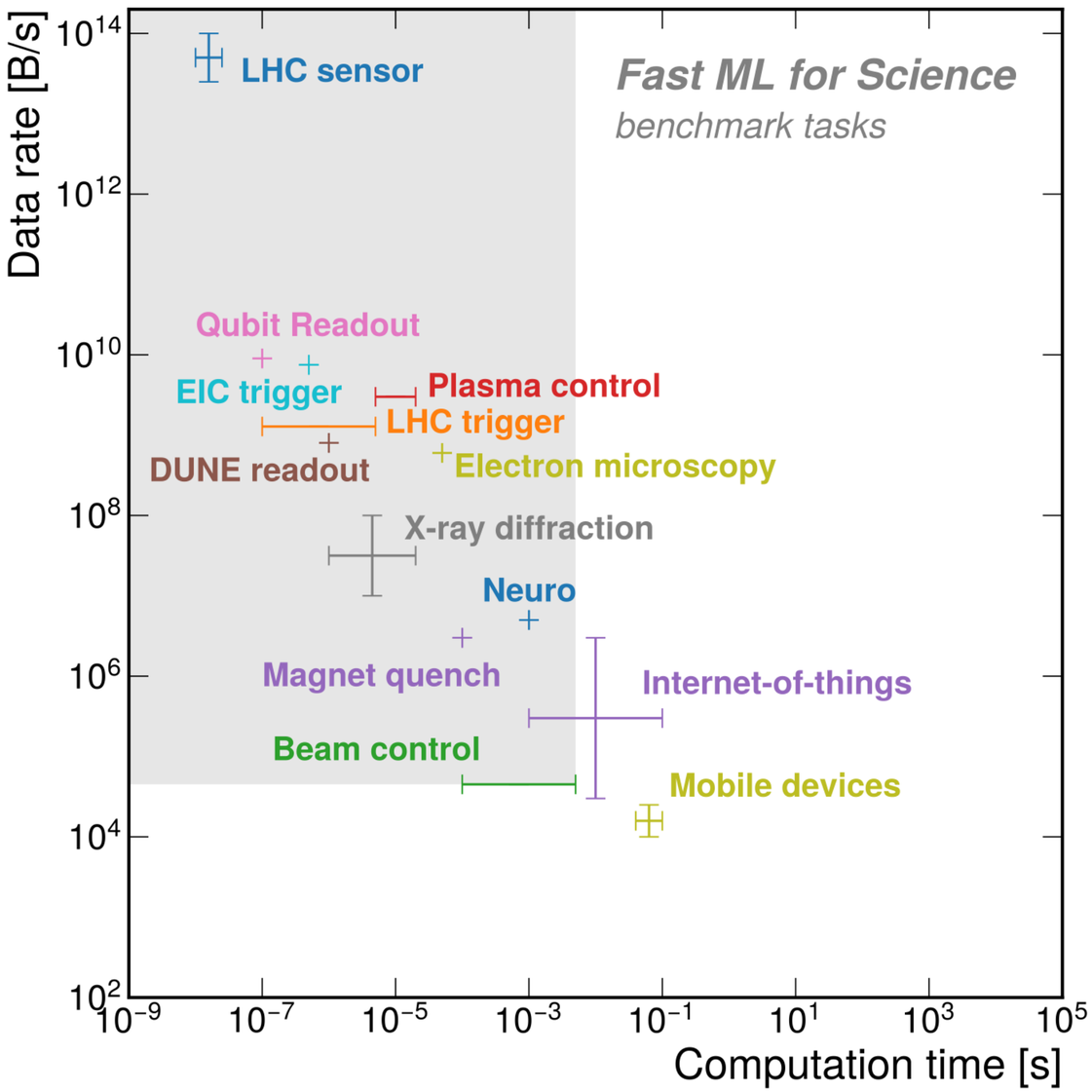


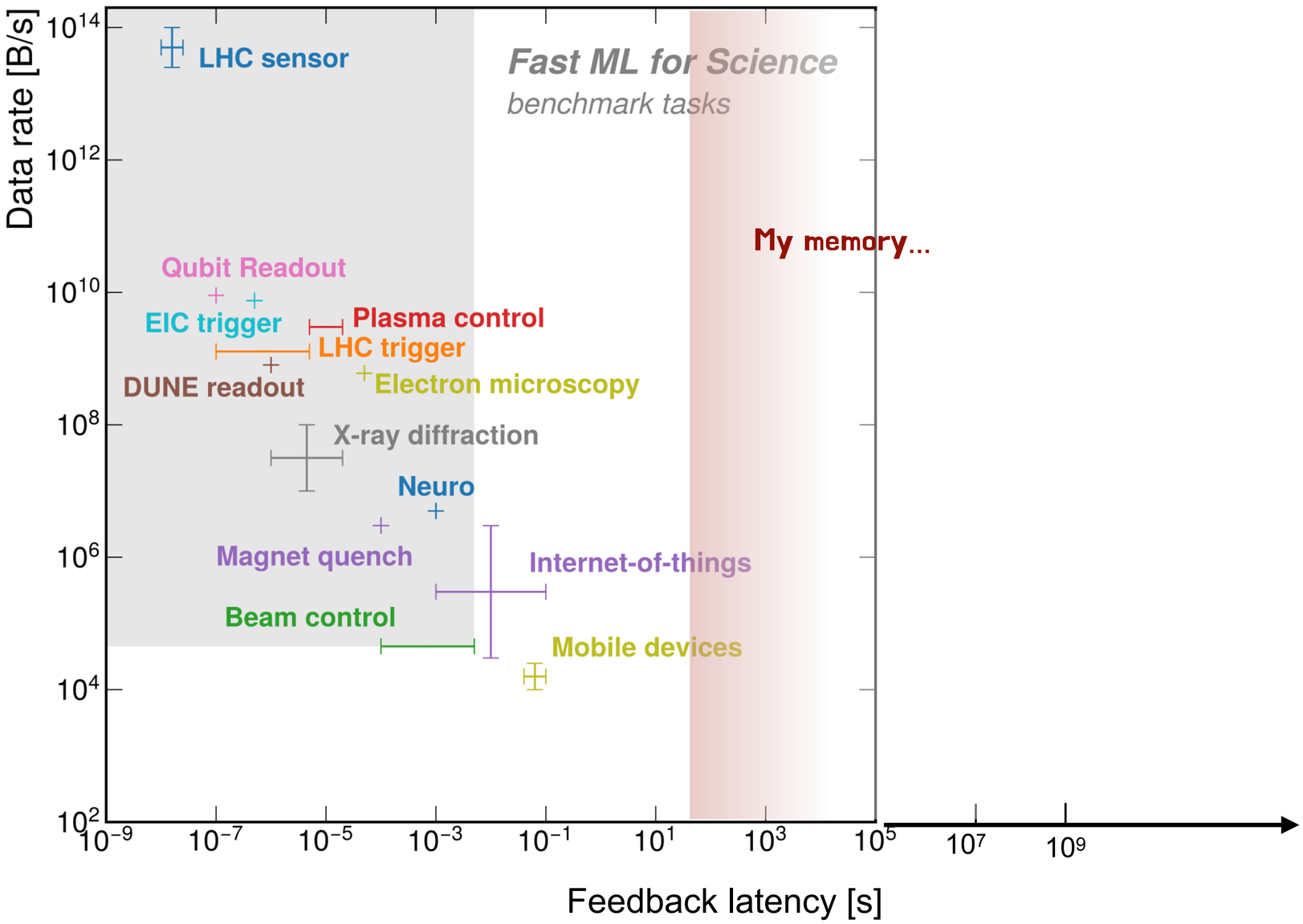
Image credit: A. Thea

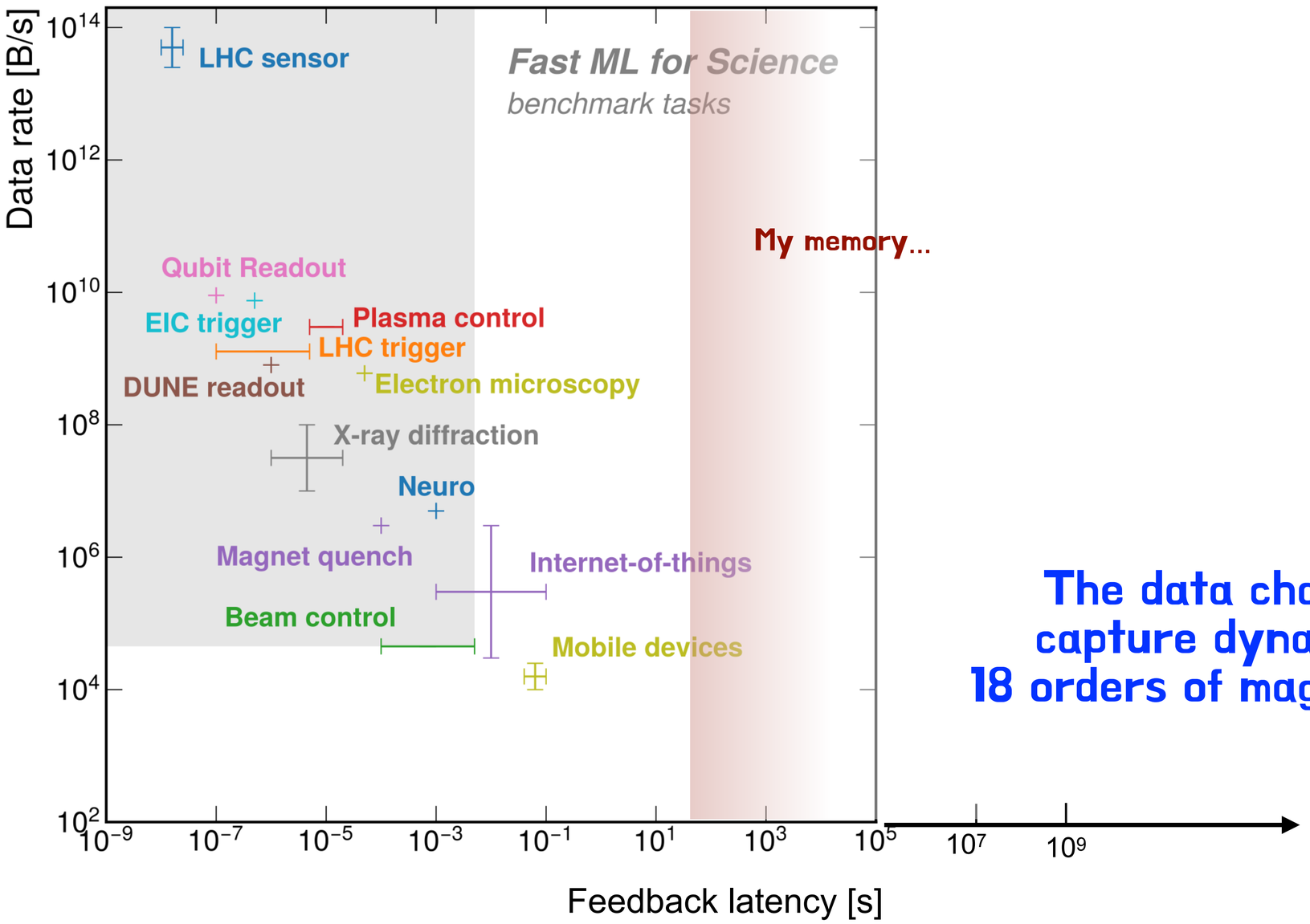
Fast and slow



- Fast control
  - Immediate response to dynamics of the experiment and data readout
  - Event timing, triggering, etc.
- Slow control
  - Detector stability over minutes, days, weeks, months,...
  - Monitoring and controlling operational parameters: electronics gains, pedestals, calibrations, etc.

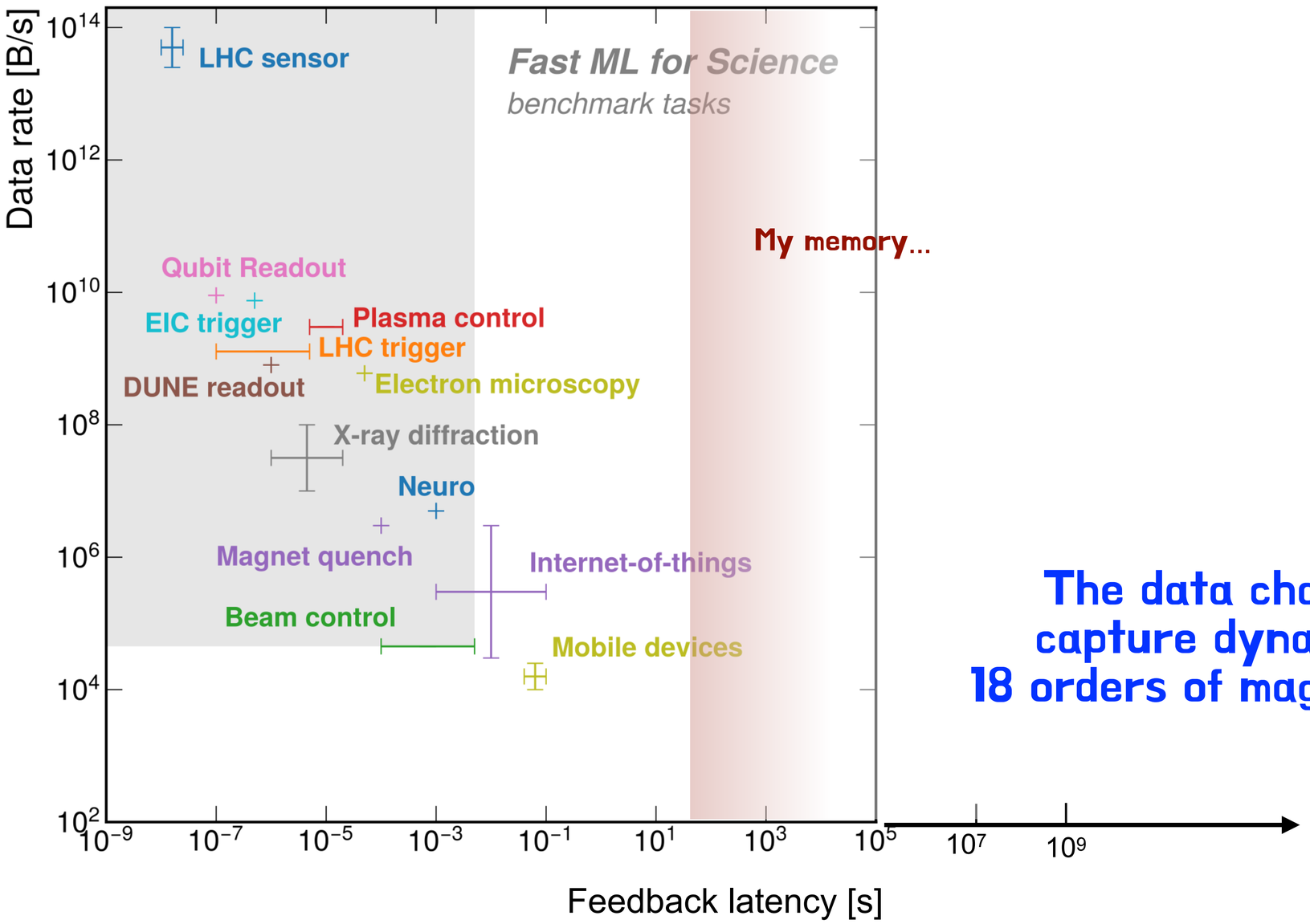

Fast ML for Science
benchmark tasks
LHC sensor
Qubit Readout
EIC trigger
Plasma control
LHC trigger
DUNE readout
Electron microscopy
X-ray diffraction
Neuro
Magnet quench
Internet-of-things
Beam control
Mobile devices
Data rate [B/s]
Computation time [s]
10^{14}
10^{12}
10^{10}
10^{8}
10^{6}
10^{4}
10^{2}
10^{-9}
10^{-7}
10^{-5}
10^{-3}
10^{-1}
10^{1}
10^{3}
10^{5}


Data rate [B/s]
Feedback latency [s]
Fast ML for Science
benchmark tasks
My memory...
LHC sensor
Qubit Readout
EIC trigger
Plasma control
LHC trigger
DUNE readout
Electron microscopy
X-ray diffraction
Neuro
Magnet quench
Internet-of-things
Beam control
Mobile devices



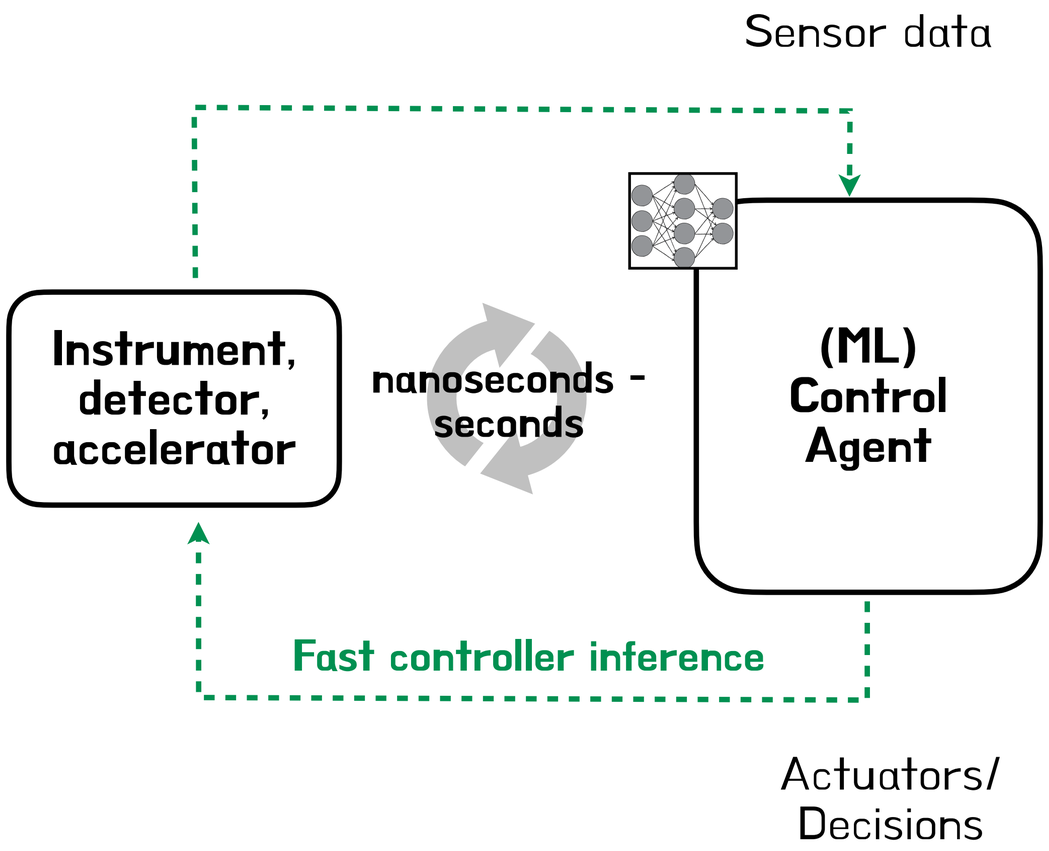
The data challenge is to capture dynamics across 18 orders of magnitude in time!

What is real-time???

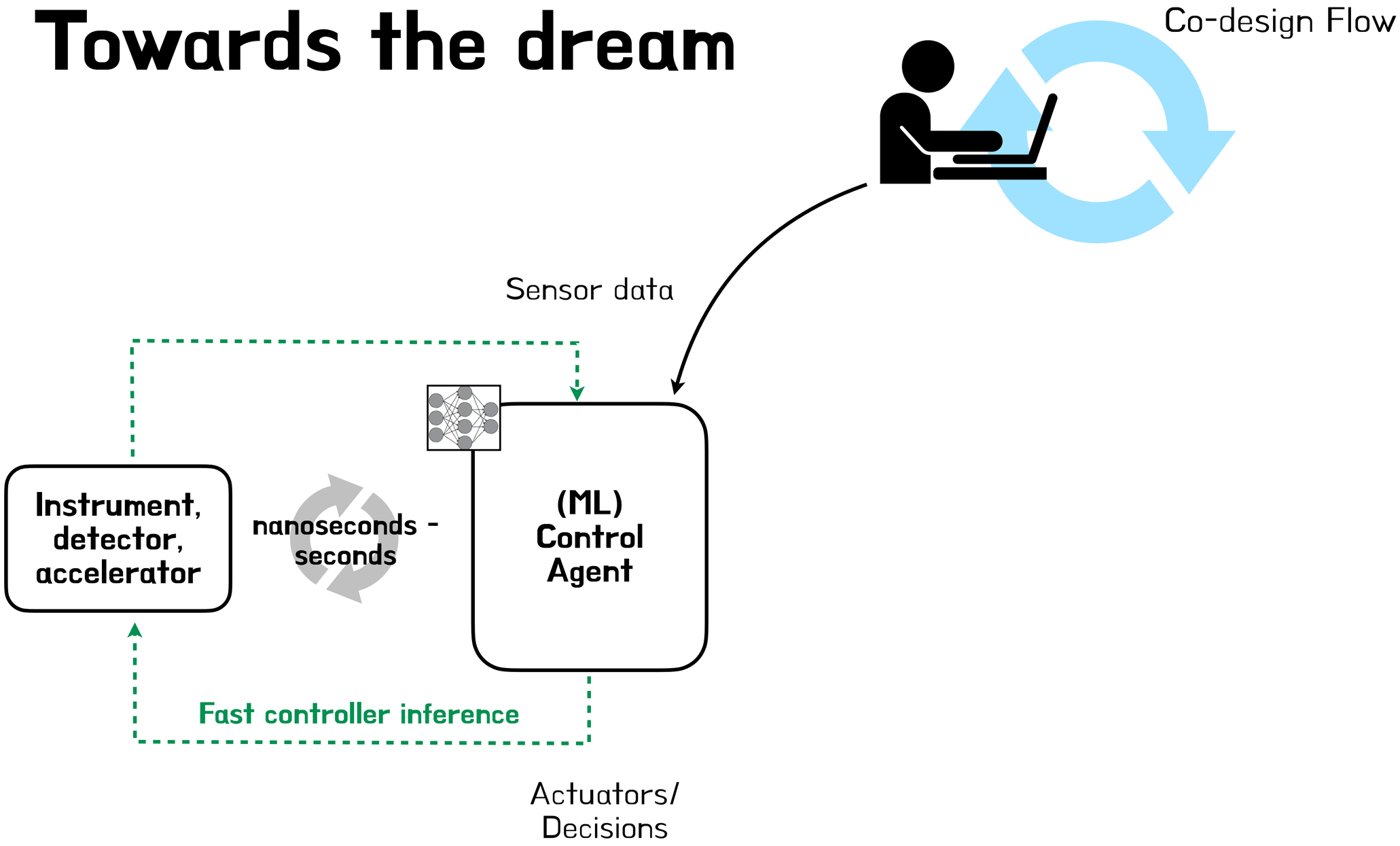


The data challenge is to capture dynamics across 18 orders of magnitude in time!

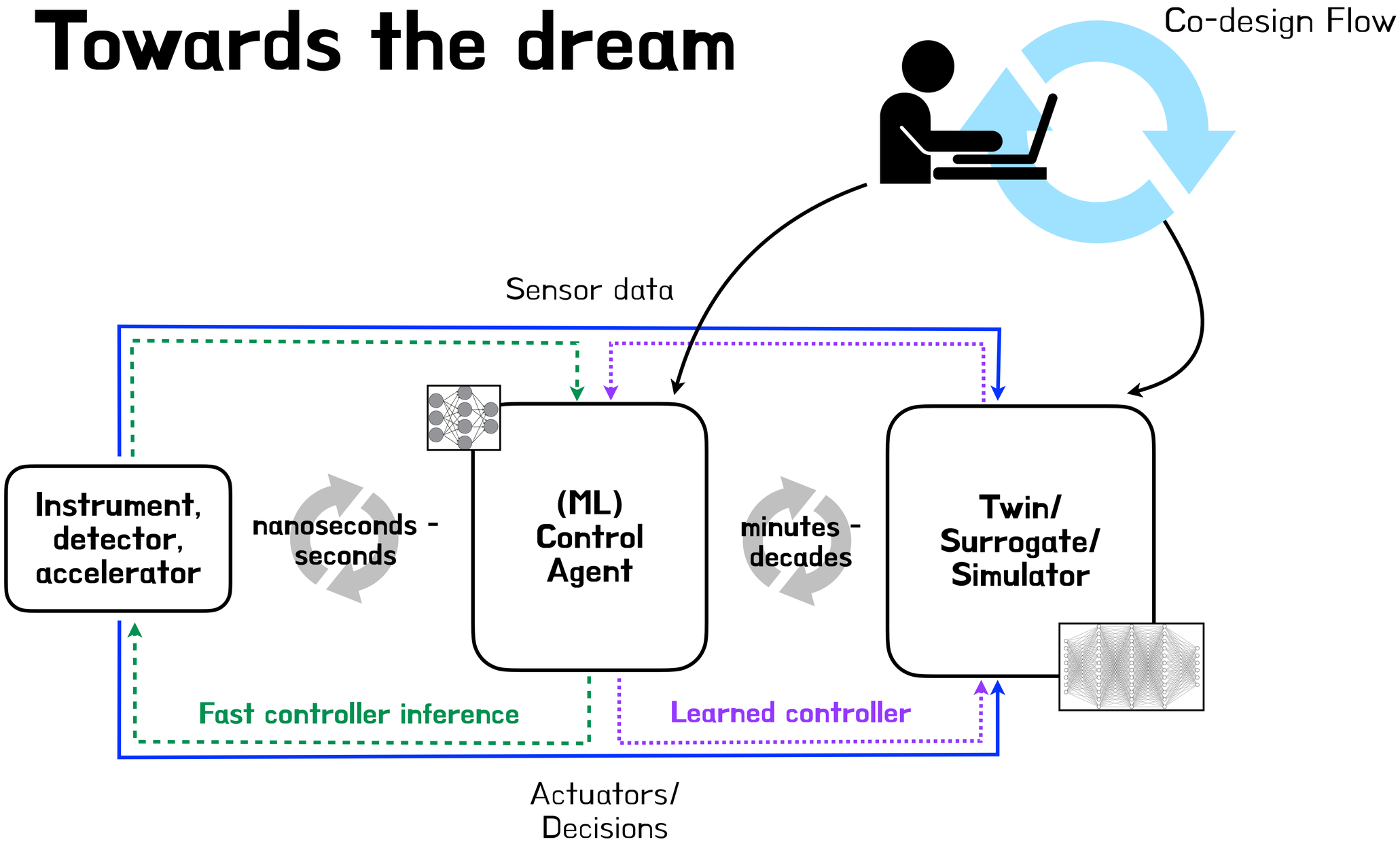
Towards the dream



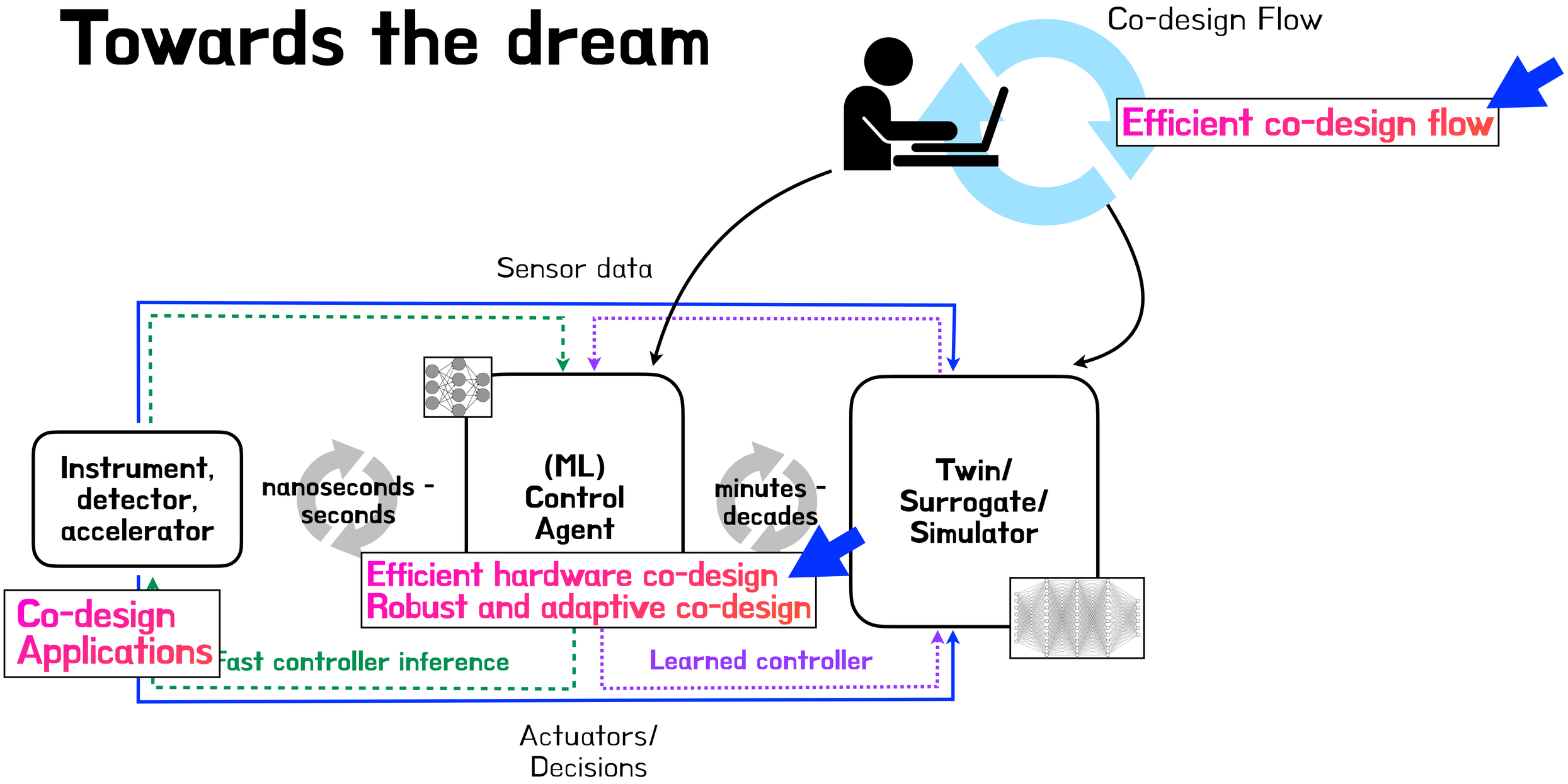
Towards the dream



Towards the dream

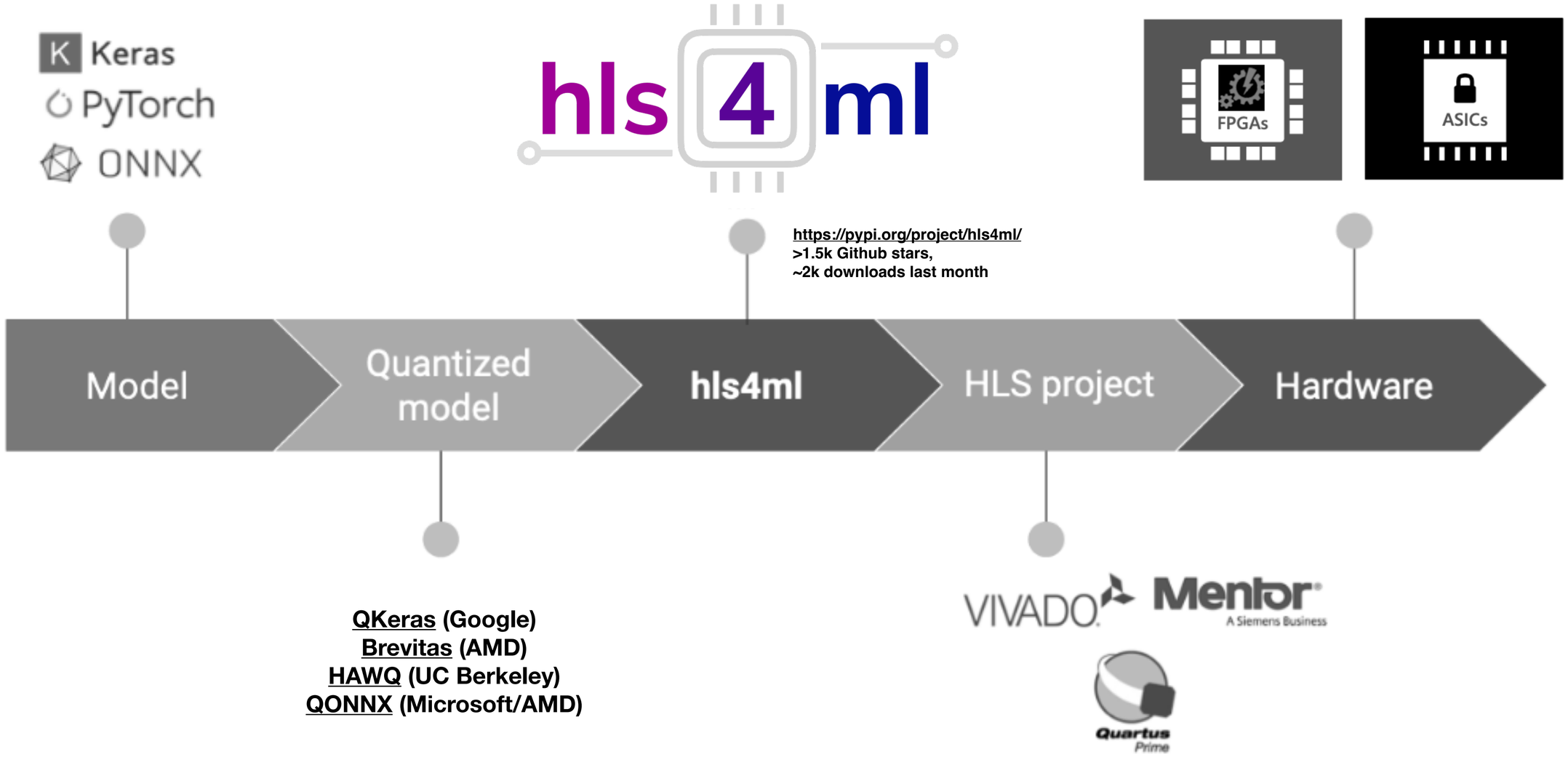


Towards the dream



One example with hls4ml

Efficient tools for efficient, embedded AI



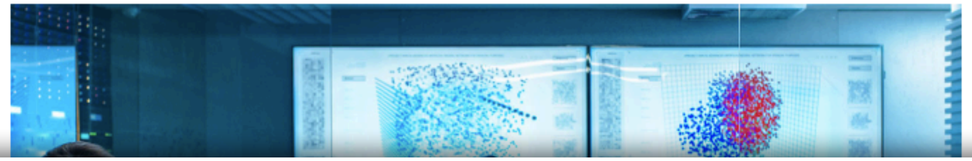
AI on ASIC

https://newsroom.sw.siemens.com/en-US/siemens-catapult-ai-nn/

PRESS RELEASE

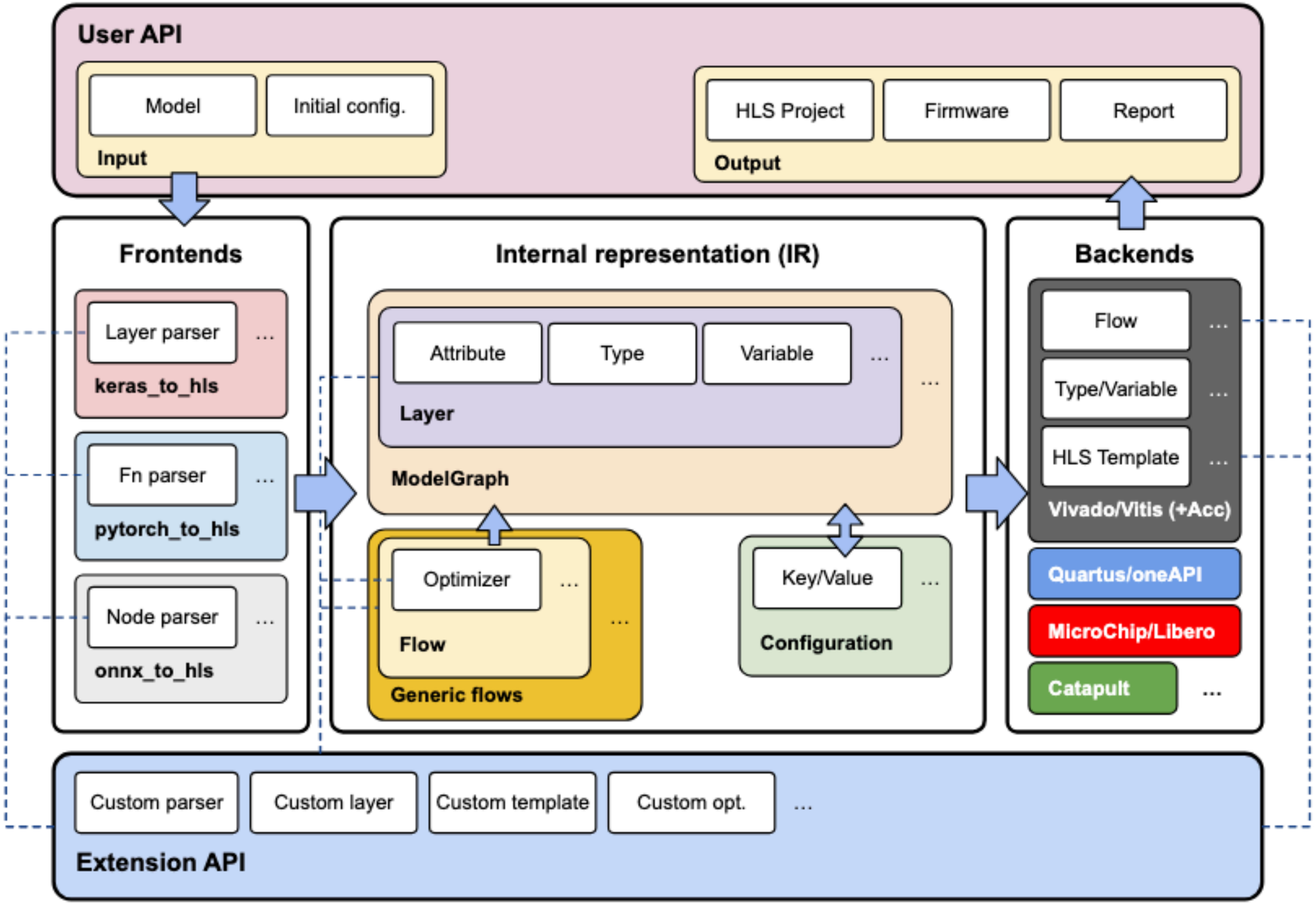
Siemens simplifies development of AI accelerators for advanced system-on-chip designs with Catapult AI NN

May 21, 2024
Plano, Texas



Catapult AI NN brings together hls4ml, an open-source package for machine learning hardware acceleration, and Siemens' Catapult™ HLS software for High-Level Synthesis. Developed in close collaboration with Fermilab, a U.S. Department of Energy Laboratory, and other leading contributors to hls4ml, Catapult AI NN addresses the unique requirements of machine learning accelerator design for power, performance, and area on custom silicon.

- Catapult AI NN offers software engineers a comprehensive solution to synthesize AI Neural Nets
- Enables software development teams to seamlessly translate AI models designed in Python into silicon-based implementations, facilitating faster and more power-efficient execution compared to standard processors

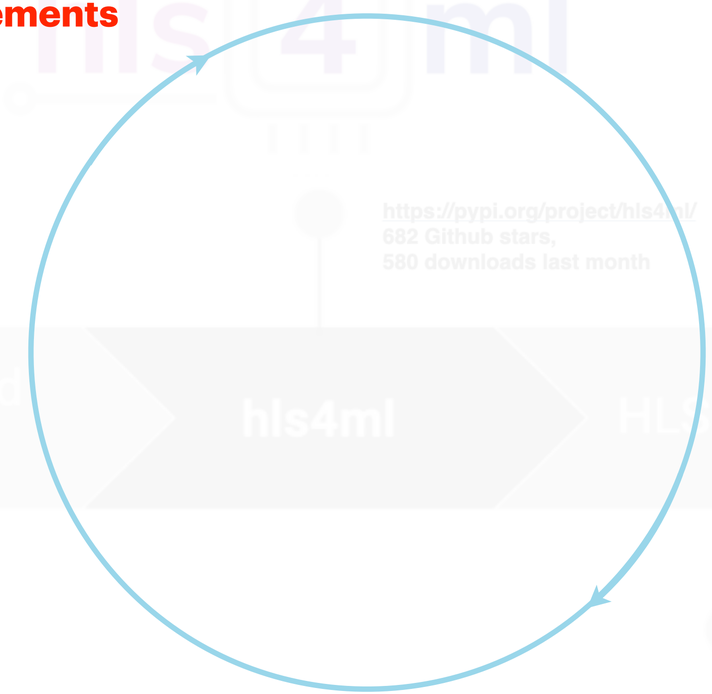

User API
Model
Initial config.
Input
HLS Project
Firmware
Report
Output
Frontends
Layer parser
...
keras_to_hls
Fn parser
...
pytorch_to_hls
Node parser
...
onnx_to_hls
Internal representation (IR)
Attribute
Type
Variable
...
Layer
...
ModelGraph
Optimizer
...
Flow
...
Generic flows
Key/Value
...
Configuration
Backends
Flow
...
Type/Variable
...
HLS Template
...
Vivado/Vitis (+Acc)
Quartus/oneAPI
MicroChip/Libero
Catapult
...
Custom parser
Custom layer
Custom template
Custom opt.
...
Extension API

Hardware - algorithm codesign

Data representation
→ ML architecture

Neural architecture search/
Hyperparameter optimization

Physics requirements



Latency?
Pipeline Interval?

Resources? Area/power?
Radiation? Cryo?

Hardware - algorithm codesign

Data representation → ML architecture

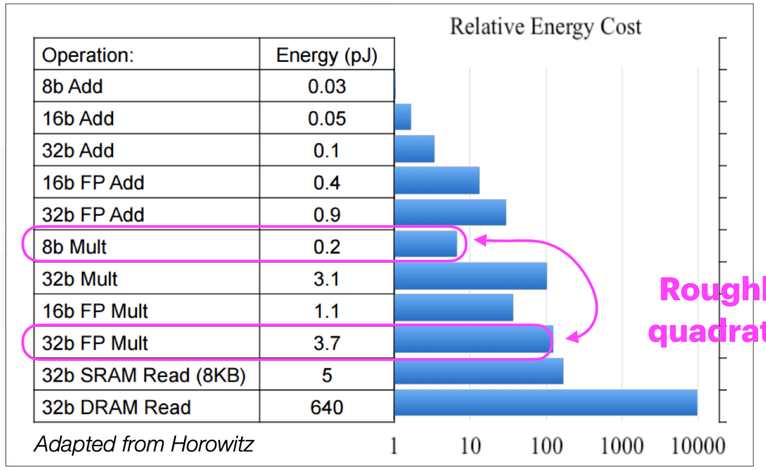
Neural architecture search/ Hyperparameter optimization

Physics requirements

Quantize network

Latency? Pipeline Interval?

Resources? Area/power? Radiation? Cryo?



Operation:	Energy (pJ)
8b Add	0.03
16b Add	0.05
32b Add	0.1
16b FP Add	0.4
32b FP Add	0.9
8b Mult	0.2
32b Mult	3.1
16b FP Mult	1.1
32b FP Mult	3.7
32b SRAM Read (8KB)	5
32b DRAM Read	640

Adapted from Horowitz

Roughly quadratic

Hardware - algorithm codesign

Data representation → ML architecture

Neural architecture search/ Hyperparameter optimization

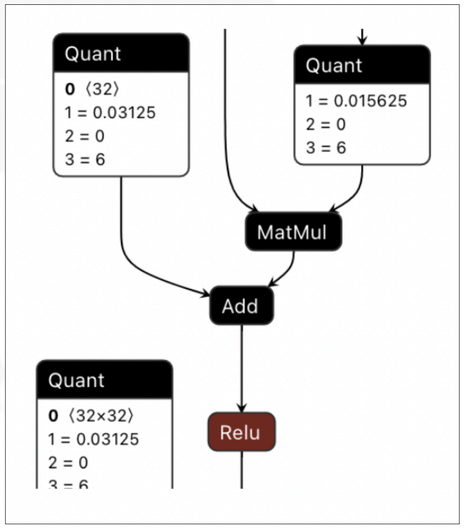
Physics requirements

Quantize network

Intermediate (quantized) representations

Latency? Pipeline Interval?

Resources? Area/power? Radiation? Cryo?



Hardware - algorithm codesign

Data representation → ML architecture

Neural architecture search/ Hyperparameter optimization

Physics requirements

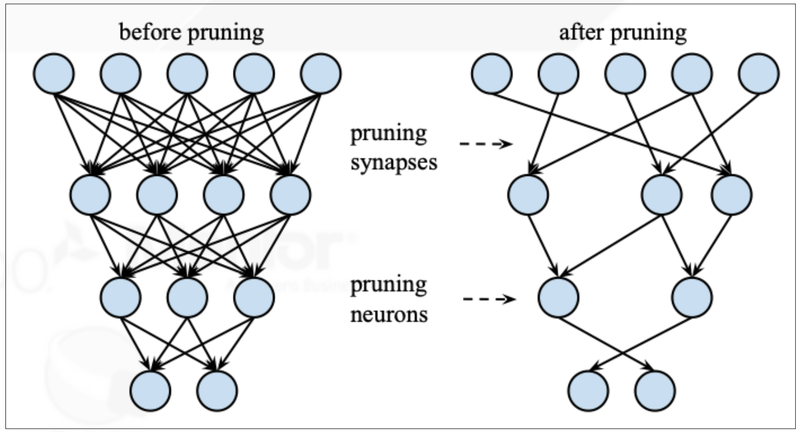
Quantize network

Intermediate (quantized) representations

Latency? Pipeline Interval?

Pruning/sparsity?

Resources? Area/power? Radiation? Cryo?



Hardware - algorithm codesign

Data representation
→ ML architecture

Neural architecture search/
Hyperparameter optimization

Physics requirements

Quantize network

Intermediate (quantized)
representations

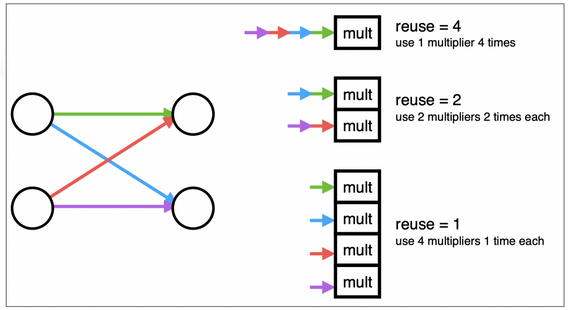
Latency?
Pipeline Interval?

Pruning/sparsity?

Resources? Area/power?
Radiation? Cryo?

Microarchitecture

Parallelization

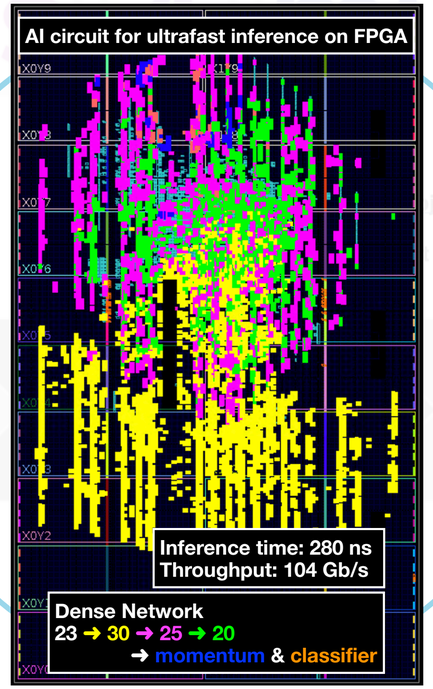


Hardware - algorithm codesign

Data representation
→ ML architecture

Neural architecture search/
Hyperparameter optimization

Physics requirements



Quantize network

Intermediate (quantized)
representations

Pruning/sparsity?

Microarchitecture

Parallelization

Synthesize, validate design,
satisfy design rules/timing

Latency?
Pipeline Interval?

Resources? Area/power?
Radiation? Cryo?

Open-source, application driven

can you help me develop design a 4 layer MLP using QKeras that can be translated to AMD FPGAs using hls4ml?

Open-source, application driven

can you help me develop design a 4 layer MLP using QKeras that can be translated to AMD FPGAs using hls4ml?

2. Design Your Quantized MLP Using QKeras

Here's how you can define a 4-layer MLP with quantized weights and activations:

python Copy code

```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from qkeras import QDense, QActivation
from qkeras.quantizers import quantized_bits, quantized_relu

# Define input and output sizes
input_size = 32   # Replace with your actual input feature size
output_size = 10  # Replace with your actual number of classes

# Define quantizers
weight_quantizer = quantized_bits(8, 0, alpha=1)        # 8-bit quantization for w
activation_quantizer = quantized_relu(8, 0)             # 8-bit quantization for a

# Build the model
model = Sequential([
    QDense(64, input_shape=(input_size,),
           kernel_quantizer=weight_quantizer,
           bias_quantizer=weight_quantizer,
           name='fc1'),
    QActivation(activation_quantizer, name='act1'),
```

Open-source, application driven

can you help me develop design a 4 layer MLP using QKeras that can be translated to AMD FPGAs using hls4ml?

2. Design Your Quantized MLP Using QKeras

Here's how you can define a 4-layer MLP with quantized weights and activations:

python Copy code

```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from qkeras import QDense, QActivation
from qkeras.quantizers import quantized_bits, quantized_relu

# Define input and output sizes
input_size = 32   # Replace with your actual input feature size
output_size = 10  # Replace with your actual number of classes

# Define quantizers
weight_quantizer = quantized_bits(8, 0, alpha=1)        # 8-bit quantization for w
activation_quantizer = quantized_relu(8, 0)             # 8-bit quantization for a

# Build the model
model = Sequential([
    QDense(64, input_shape=(input_size,),
           kernel_quantizer=weight_quantizer,
           bias_quantizer=weight_quantizer,
           name='fc1'),
    QActivation(activation_quantizer, name='act1'),
```

3. Train Your Model

4. Prepare for Conversion with hls4ml

5. Convert the Model Using hls4ml

Specify the FPGA part number corresponding to your AMD FPGA. Since AMD acquired Xilinx, you can use Xilinx part numbers.

python Copy code

```python
hls_model = hls4ml.converters.convert_from_keras_model(
    model,
    hls_config=config,
    output_dir='hls4ml_prj',
    part='xcvu9p-flgb2104-2-i'  # Replace with your specific FPGA part number
)
```

Open-source, application driven

can you help me develop design a 4 layer MLP using QKeras that can be translated to AMD FPGAs using hls4ml?

2. Design Your Quantized MLP Using QKeras

Here's how you can define a 4-layer MLP with quantized weights and activations:

python Copy code

```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from qkeras import QDense, QActivation
from qkeras.quantizers import quantized_bits, quantized_relu

# Define input and output sizes
input_size = 32   # Replace with your actual input feature size
output_size = 10  # Replace with your actual number of classes

# Define quantizers
weight_quantizer = quantized_bits(8, 0, alpha=1)        # 8-bit quantization for w
activation_quantizer = quantized_relu(8, 0)             # 8-bit quantization for a

# Build the model
model = Sequential([
    QDense(64, input_shape=(input_size,),
           kernel_quantizer=weight_quantizer,
           bias_quantizer=weight_quantizer,
           name='fc1'),
    QActivation(activation_quantizer, name='act1'),
```

3. Train Your Model

4. Prepare for Conversion with hls4ml

5. Convert the Model Using hls4ml

Specify the FPGA part number corresponding to your AMD FPGA. Since AMD acquired Xilinx, you can use Xilinx part numbers.

python Copy code

```python
hls_model = hls4ml.converters.convert_from_keras_model(
    model,
    hls_config=config,
    output_dir='hls4ml_prj',
    part='xcvu9p-flgb2104-2-i'  # Replace with your specific FPGA part number
)
```

6. Compile and Build the HLS Model

7. Validate the HLS Model

Compare the outputs of the original Keras model and the HLS model to ensure consistency.

python Copy code

```python
import numpy as np

# Generate some test data
X_test = np.random.rand(100, input_size)
```

8. Synthesize the Design Using Vivado HLS

Open-source, application driven

can you help me develop design a 4 layer MLP using QKeras that can be translated to AMD FPGAs using hls4ml?

2. Design Your Quantized MLP Using QKeras

Here's how you can define a 4-layer MLP with quantized weights and activations:

python Copy code

```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from qkeras import QDense, QActivation
from qkeras.quantizers import quantized_bits, quantized_relu

# Define input and output sizes
input_size = 32   # Replace with your actual input feature size
output_size = 10  # Replace with your actual number of classes

# Define quantizers
weight_quantizer = quantized_bits(8, 0, alpha=1)        # 8-bit quantization for w
activation_quantizer = quantized_relu(8, 0)             # 8-bit quantization for a

# Build the model
model = Sequential([
    QDense(64, input_shape=(input_size,),
           kernel_quantizer=weight_quantizer,
           bias_quantizer=weight_quantizer,
           name='fc1'),
    QActivation(activation_quantizer, name='act1'),
```

3. Train Your Model

4. Prepare for Conversion with hls4ml

5. Convert the Model Using hls4ml

Specify the FPGA part number corresponding to your AMD FPGA. Since AMD acquired Xilinx, you can use Xilinx part numbers.

python Copy code

```python
hls_model = hls4ml.converters.convert_from_keras_model(
    model,
    hls_config=config,
    output_dir='hls4ml_prj',
    part='xcvu9p-flgb2104-2-i'  # Replace with your specific FPGA part number
)
```

6. Compile and Build the HLS Model

7. Validate the HLS Model

Compare the outputs of the original Keras model and the HLS model to ensure consistency.

python Copy code

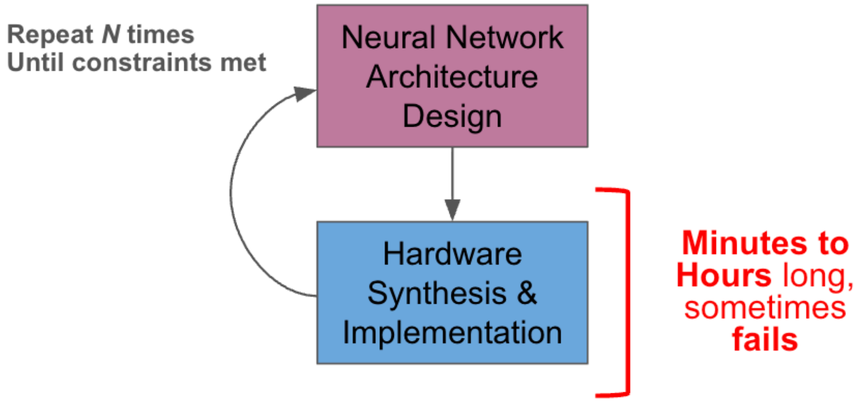
```python
import numpy as np

# Generate some test data
X_test = np.random.rand(100, input_size)
```

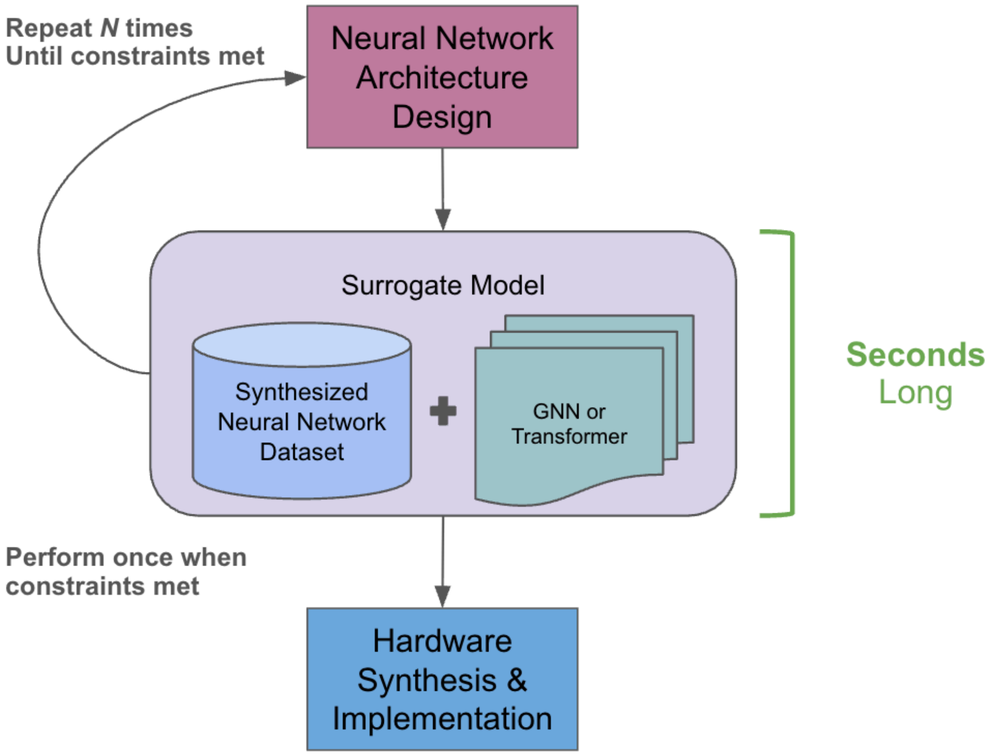
8. Synthesize the Design Using Vivado HLS

wa-hls4ml — an hls4ml surrogate model

Traditional Codesign Workflow



Proposed Codesign Workflow



Hardware - algorithm codesign

https://arxiv.org/abs/2501.05515
Tutorial at the Fast ML for Science 2025

Data representation
→ ML architecture

Neural architecture search/
Hyperparameter optimization

Physics requirements

Quantize network

Intermediate (quantized)
representations

Latency?
Pipeline Interval?

Pruning/sparsity?

Resources? Area/power?
Radiation? Cryo?

Microarchitecture

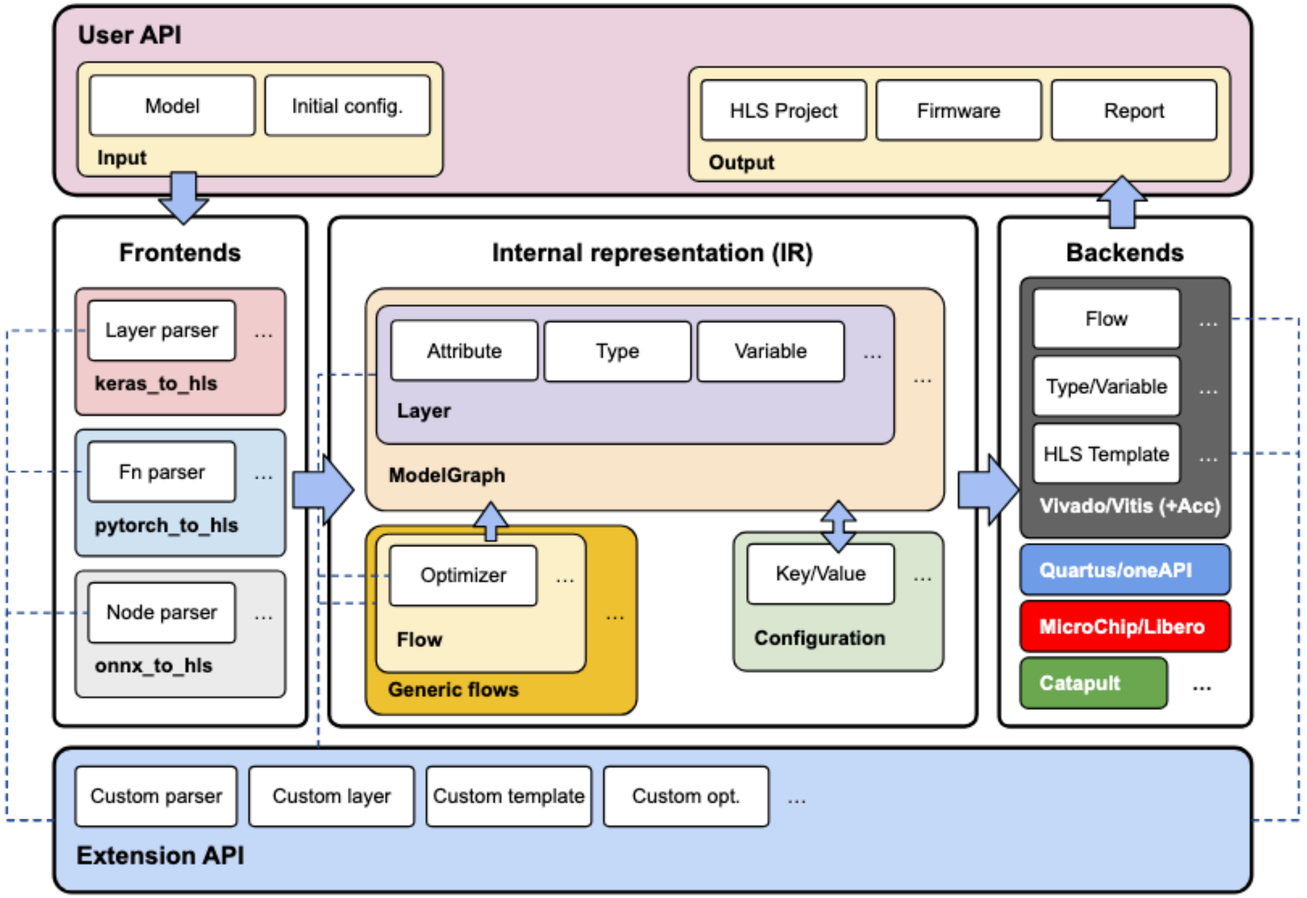
Synthesize, validate design,
satisfy design rules/timing

Parallelization

Surrogate Neural Architecture Co-design Package (SNAC-Pack) powered with wa-hls4ml

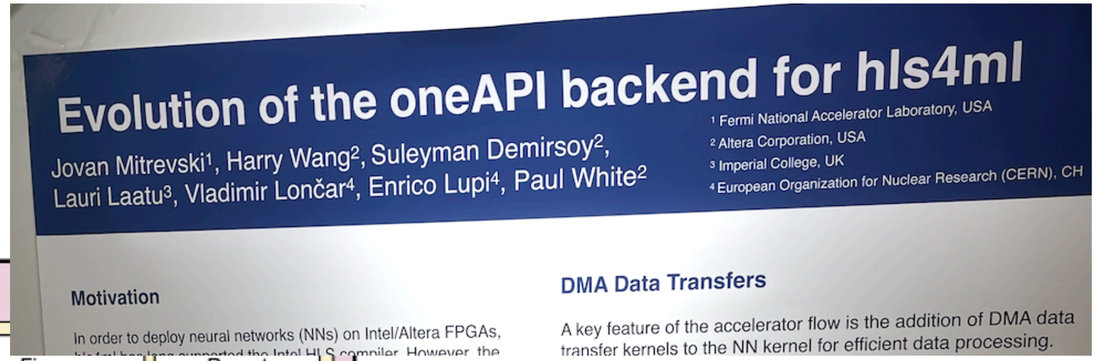
J. Vreca, J. Mitrevski, A Girjoaba, K Govoroka

Backends



J. Vreca, J. Mitrevski, A Girjoaba, K Govoroka

Backends



Evolution of the oneAPI backend for hls4ml

Jovan Mitrevski[1], Harry Wang[2], Suleyman Demirsoy[2], Lauri Laatu[3], Vladimir Lončar[4], Enrico Lupi[4], Paul White[2]

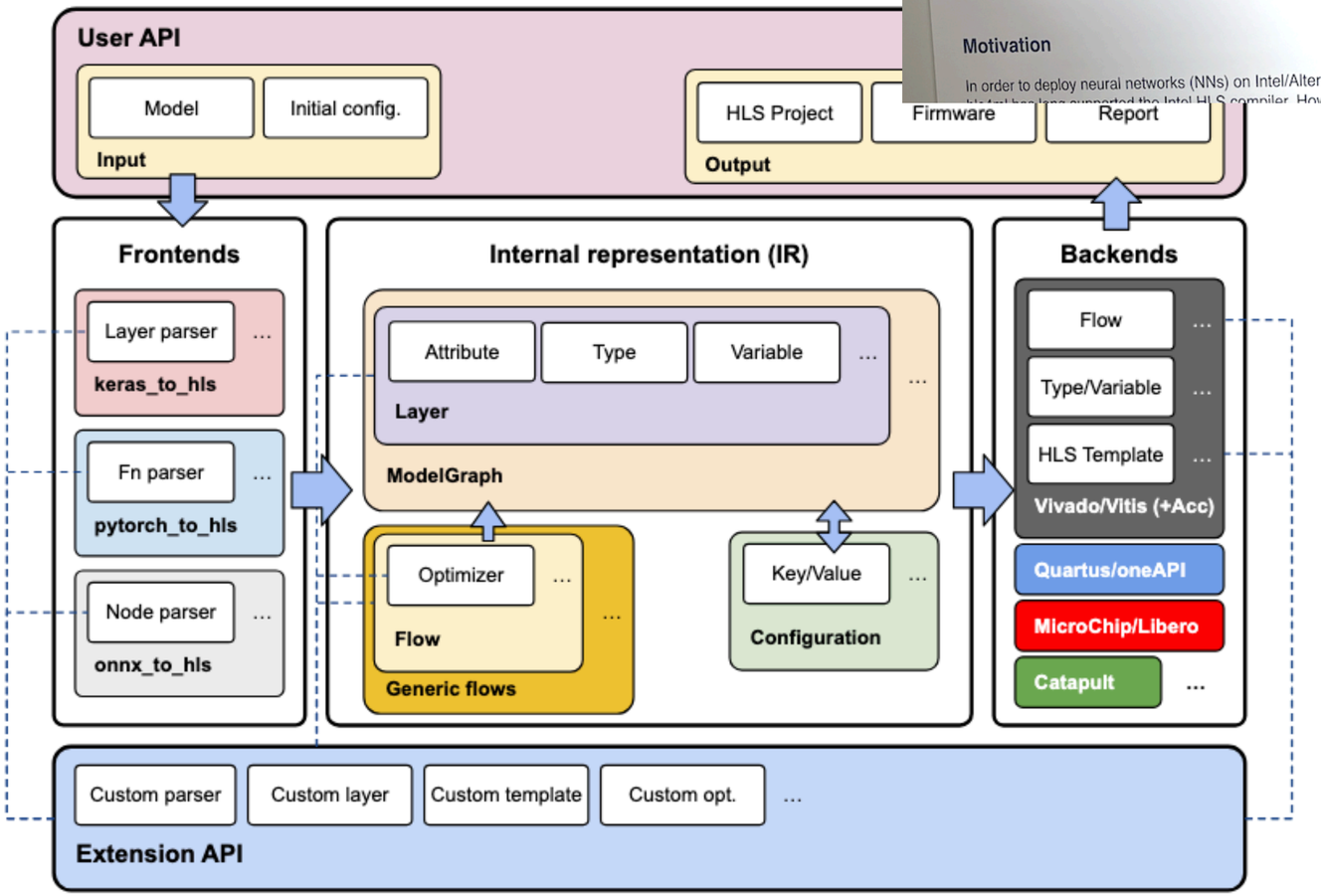
[1] Fermi National Accelerator Laboratory, USA
[2] Altera Corporation, USA
[3] Imperial College, UK
[4] European Organization for Nuclear Research (CERN), CH

Motivation

In order to deploy neural networks (NNs) on Intel/Altera FPGAs,

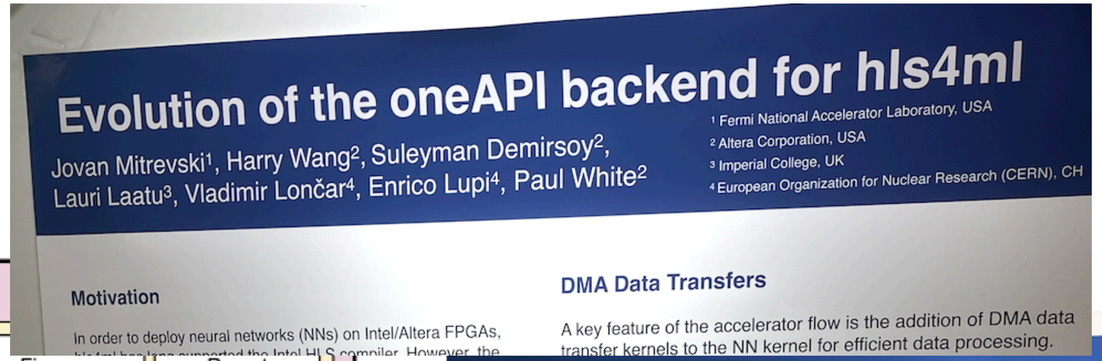
DMA Data Transfers

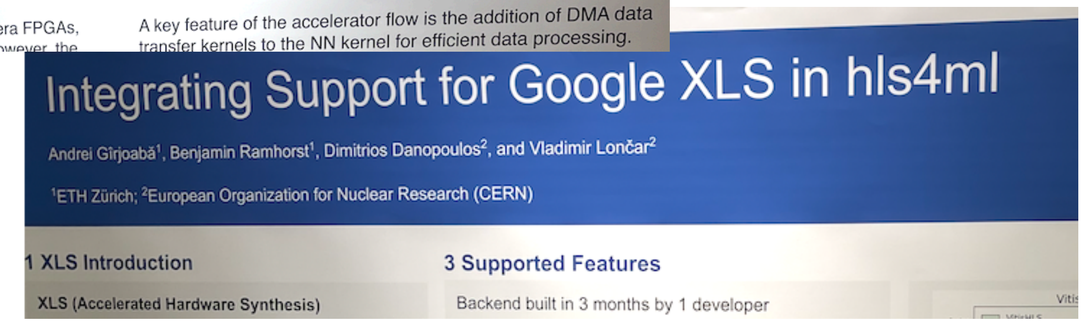
A key feature of the accelerator flow is the addition of DMA data transfer kernels to the NN kernel for efficient data processing.

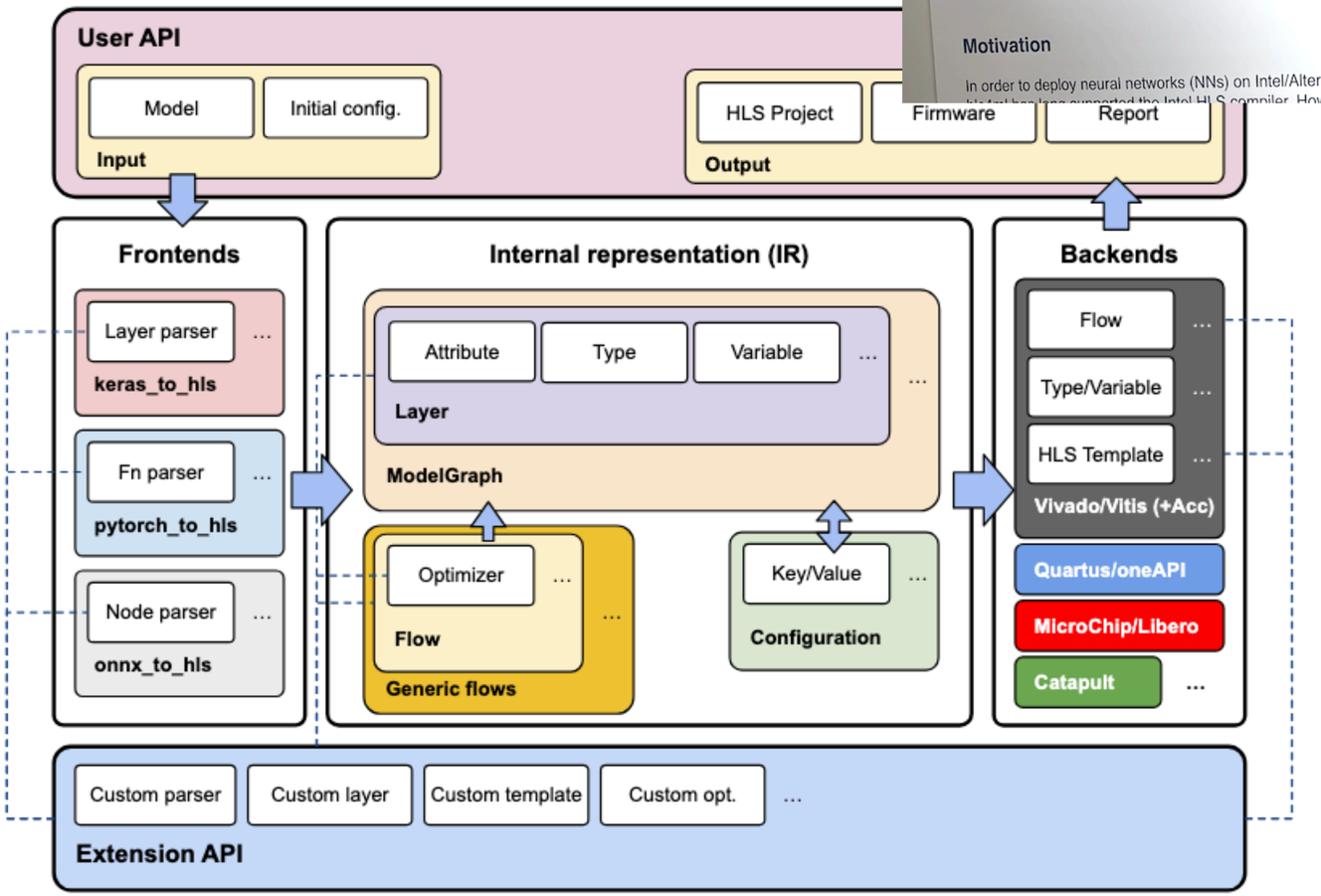


J. Vreca, J. Mitrevski, A Girjoaba, K Govoroka

Backends

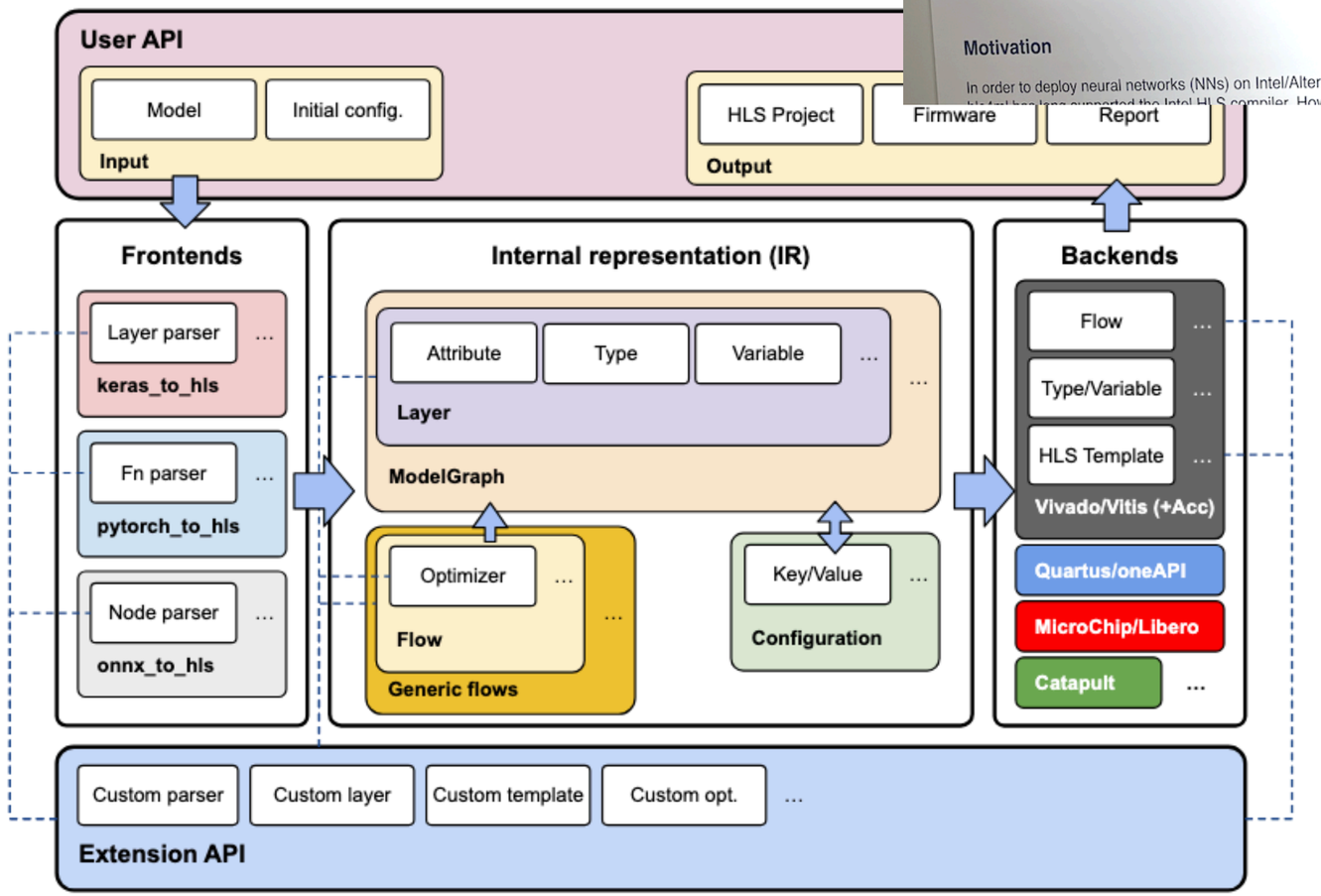


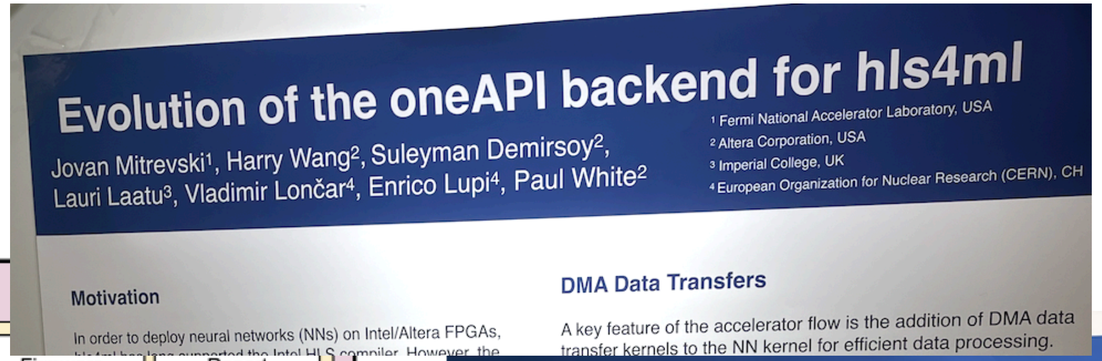


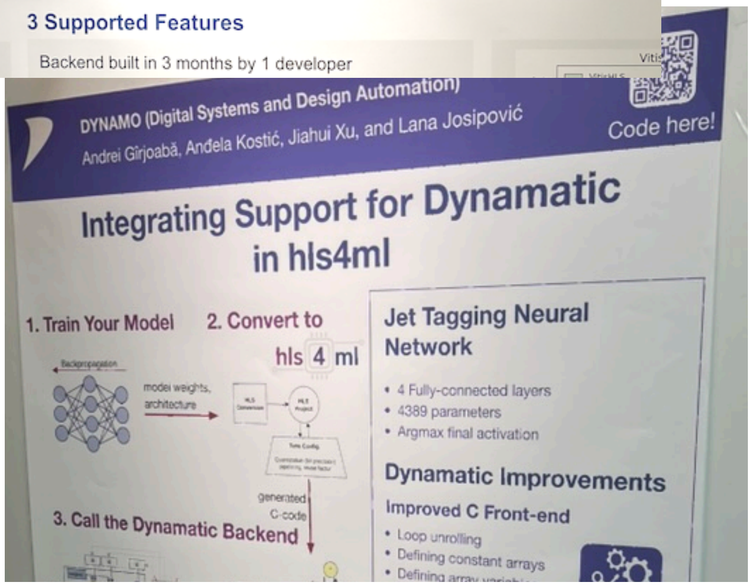


J. Vreca, J. Mitrevski, A Girjoaba, K Govoroka

Backends

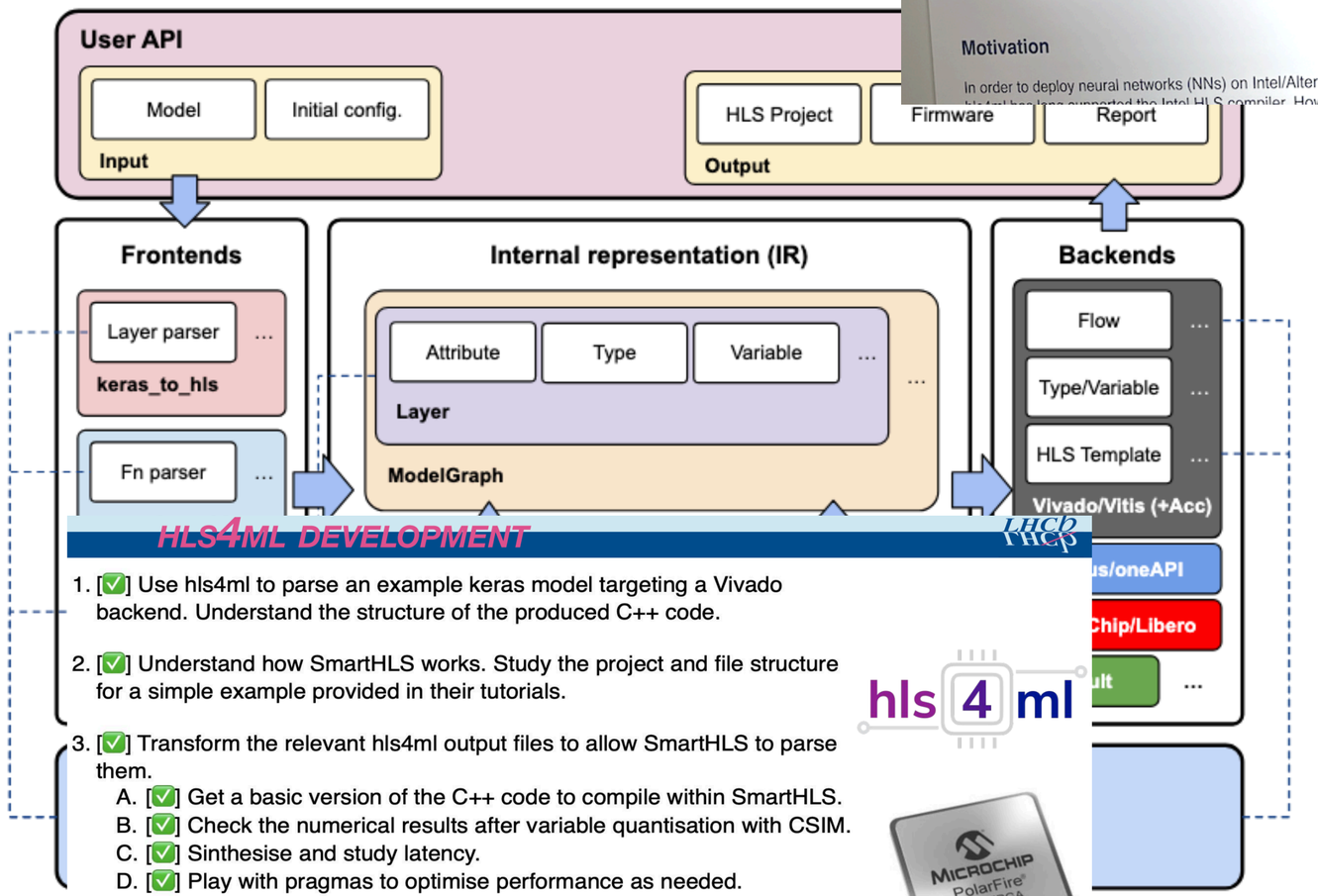


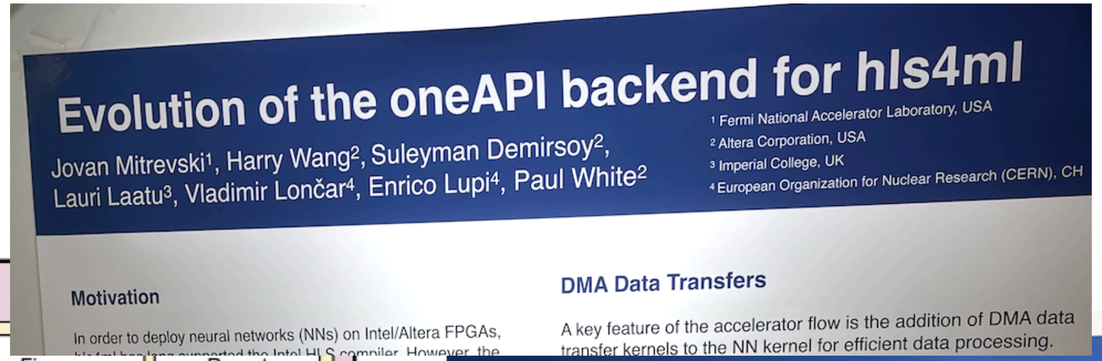


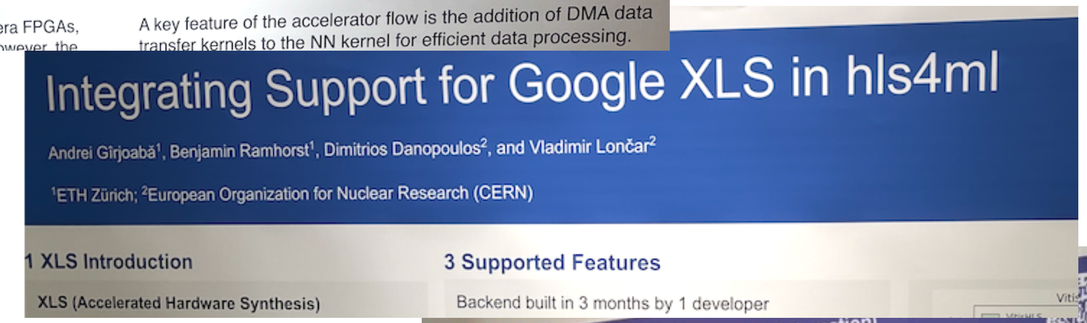


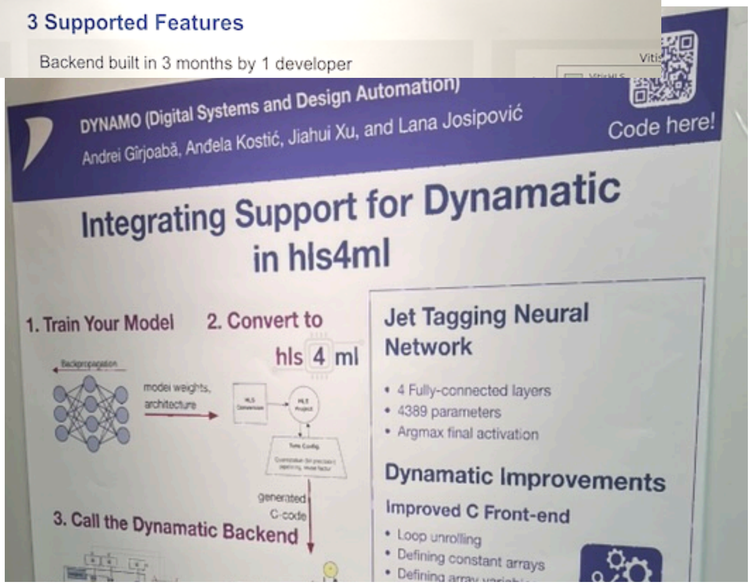
J. Vreca, J. Mitrevski, A Girjoaba, K Govoroka

Backends









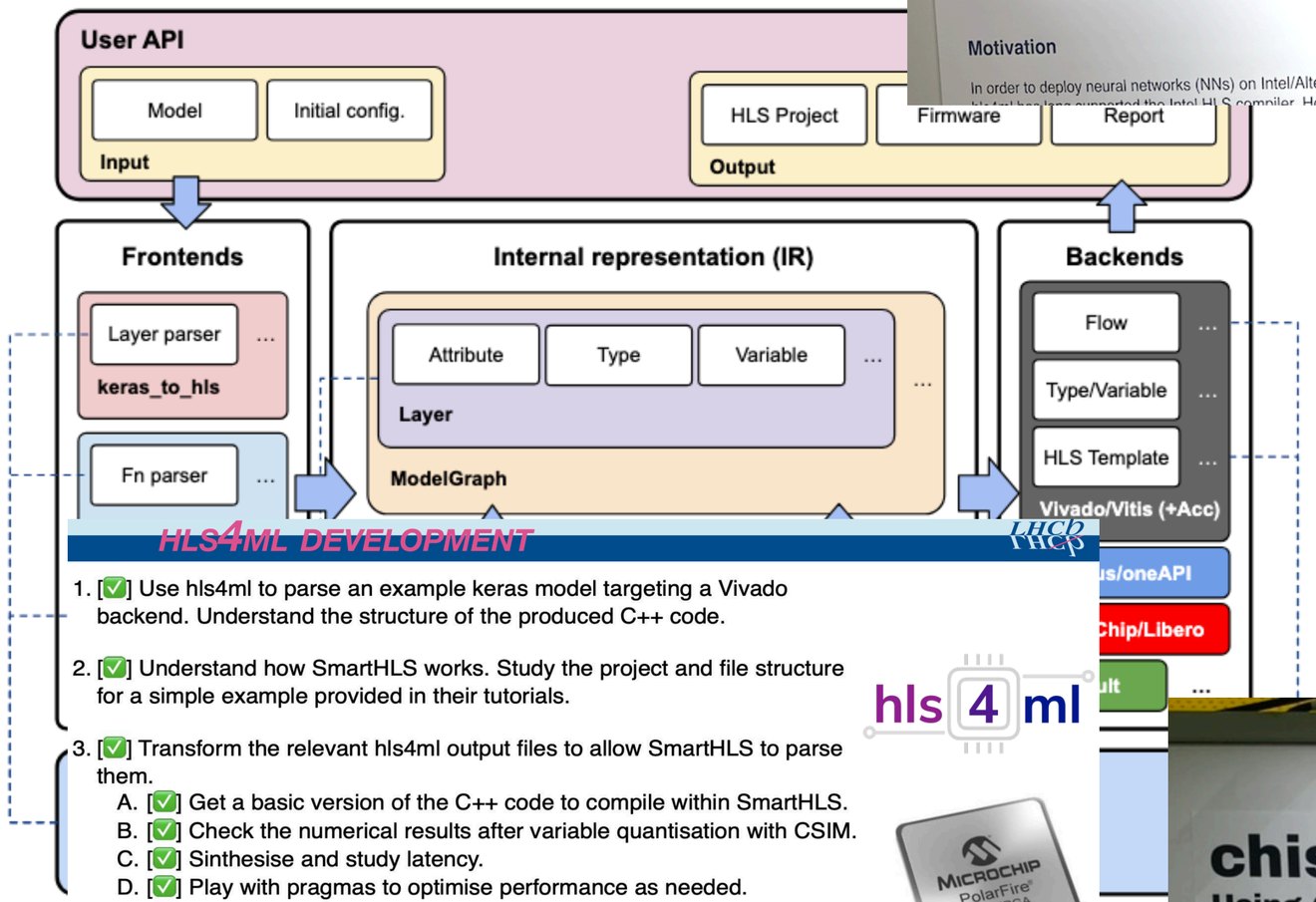
HLS4ML DEVELOPMENT

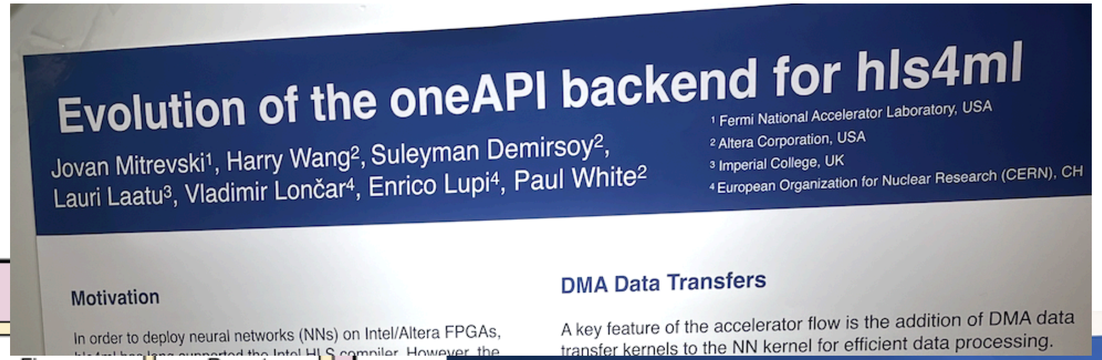
1. [✅] Use hls4ml to parse an example keras model targeting a Vivado backend. Understand the structure of the produced C++ code.
2. [✅] Understand how SmartHLS works. Study the project and file structure for a simple example provided in their tutorials.
3. [✅] Transform the relevant hls4ml output files to allow SmartHLS to parse them.
   A. [✅] Get a basic version of the C++ code to compile within SmartHLS.
   B. [✅] Check the numerical results after variable quantisation with CSIM.
   C. [✅] Sinthesise and study latency.
   D. [✅] Play with pragmas to optimise performance as needed.
4. [✅] Abstract from the learnings and implement an expansion of hls4ml to cover this new backend.
5. [✅] Use the new expansion of hls4ml to convert the LHCb ML models.

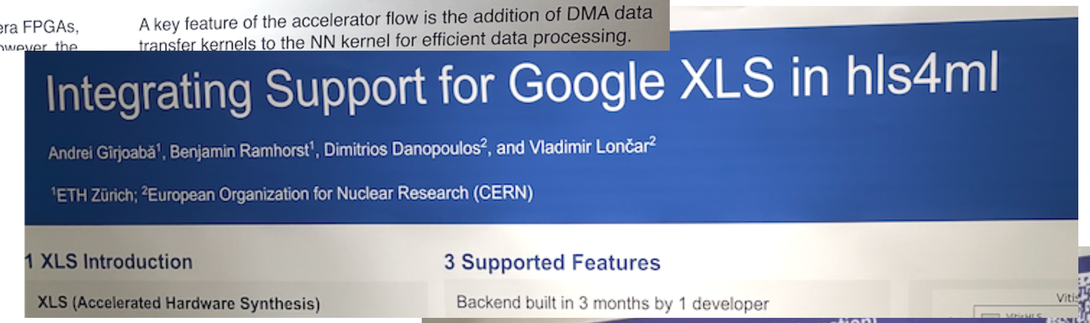
https://github.com/vloncar/hls4ml/tree/libero_backend

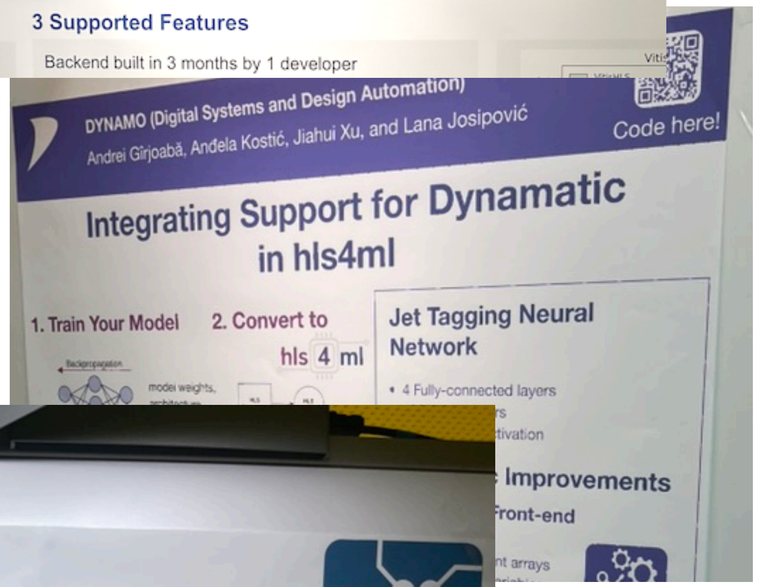
J. Vreca, J. Mitrevski, A Girjoaba, K Govoroka

Backends





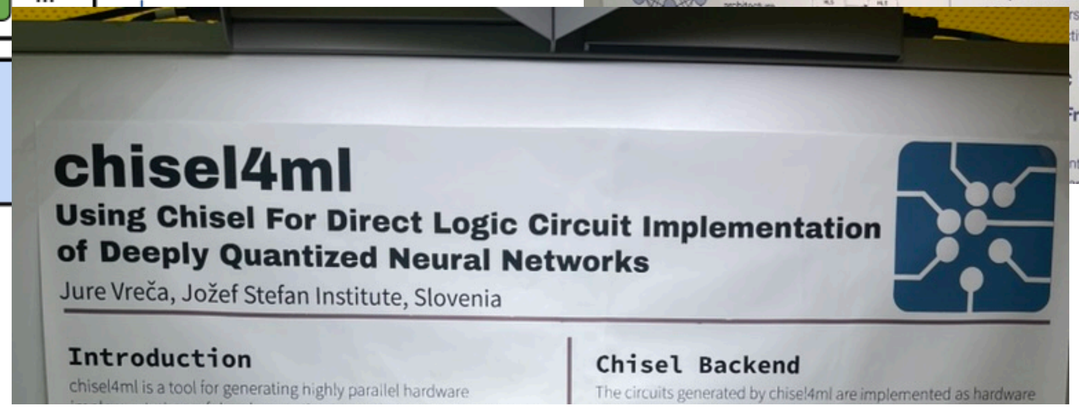




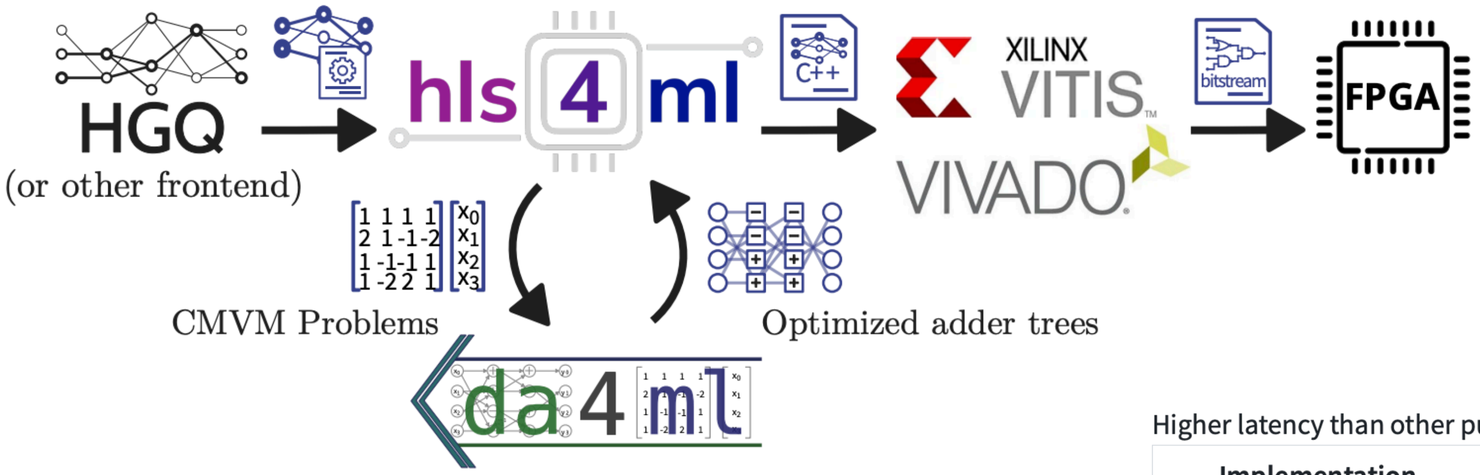
HLS4ML DEVELOPMENT

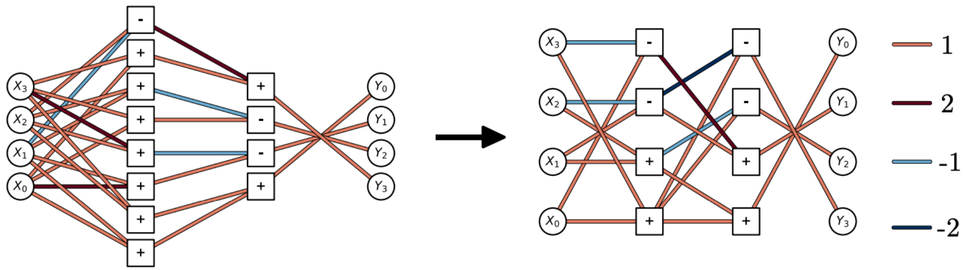
1. [✅] Use hls4ml to parse an example keras model targeting a Vivado backend. Understand the structure of the produced C++ code.
2. [✅] Understand how SmartHLS works. Study the project and file structure for a simple example provided in their tutorials.
3. [✅] Transform the relevant hls4ml output files to allow SmartHLS to parse them.
   A. [✅] Get a basic version of the C++ code to compile within SmartHLS.
   B. [✅] Check the numerical results after variable quantisation with CSIM.
   C. [✅] Sinthesise and study latency.
   D. [✅] Play with pragmas to optimise performance as needed.
4. [✅] Abstract from the learnings and implement an expansion of hls4ml to cover this new backend.
5. [✅] Use the new expansion of hls4ml to convert the LHCb ML models.

https://github.com/vloncar/hls4ml/tree/libero_backend



da4ml



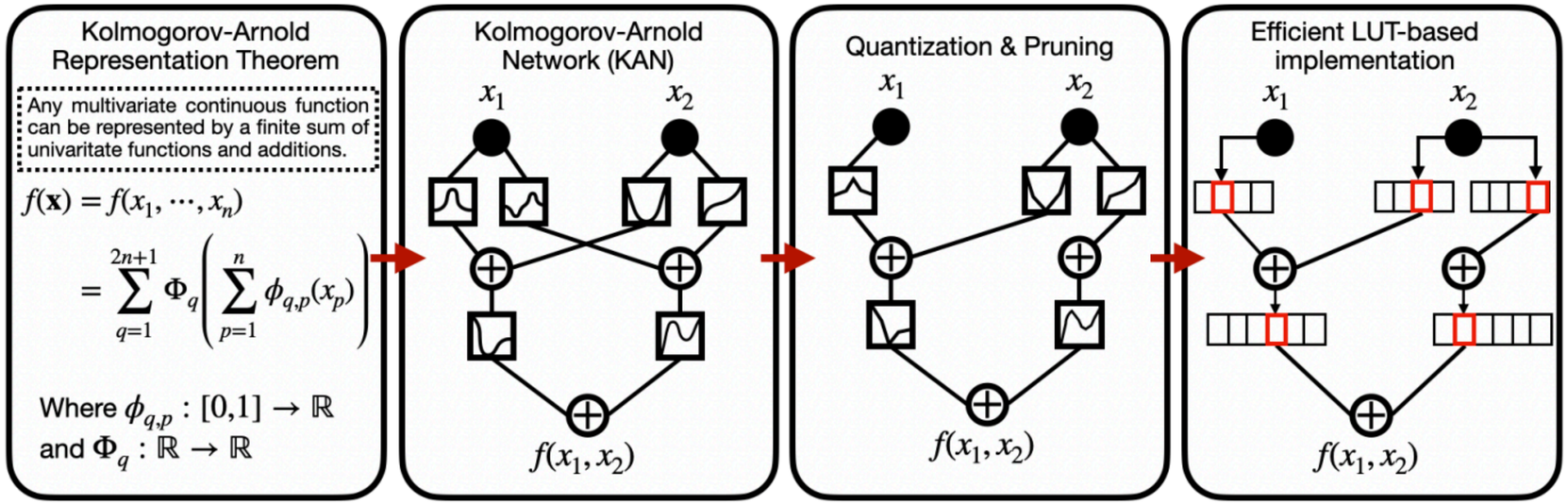


Higher latency than other pure LUT-based methods, but similar LUT usage and can achieve higher accuracy.

Implementation	Accuracy	Latency [ns]	LUT	DSP	FF	F_{max} [MHz]	II [cc]
`HGQ+hls4ml+DA`	76.9%	44.1	12,682	0	19,056	702.	1
`HGQ+da4ml (RTL)`	76.5%	23.1	6,165	0	7,207	736.	1
`HGQ+hls4ml`	76.9%	57.6	16,081	57	26,484	729.	1
`HGQ+hls4ml`	76.5%	67.2	8,548	30	14,418	521.	1
`QKeras+hls4ml` [1]	76.3%	105	5,504	175	3,036	143.	2
`DWN` [2]	76.3%	14.4	6,302	0	4,128	695.	1
`MetaML-Pro` [3]	76.1%	50	13,042	70	N/A	200	1
`NeuraLUT-Assemble` [4]	76.0%	2.1	1,780	0	540	940.	1
`TreeLUT` [5]	75.6%	2.7	2,234	0	347	735.	1

KANs

D. Hoang



Dataset	Model	Accuracy (%)	LUT	FF	DSP	BRAM	F_{max} (MHz)	Latency (ns)	Area×Delay (LUT×ns)
MNIST	**KAN-Quantized-Pruned**	96.1	**1323**	**546**	0	0	316	9.6	1.27×10^4
	NeuraLUT-Assemble [4]	**97.9**	5070	725	0	0	863	**2.1**	$\mathbf{1.06 \times 10^4}$
	TreeLUT [16]	96.6	4478	597	0	0	791	2.5	1.12×10^4
	DWN [5]	97.8	2092	1757	0	0	873	9.2	1.92×10^4
	PolyLUT-Add [20]	96.0	14810	2609	0	0	625	10	1.48×10^5
	AmigoLUT-NeuraLUT [34]	95.5	16081	13292	0	0	**925**	7.6	1.22×10^5
	NeuraLUT [3]	96.0	54798	3757	0	0	431	12	6.58×10^5
	PolyLUT [2]	97.5	75131	4668	0	0	353	17	1.38×10^6
	FINN [30]	96.0	91131	—	0	5	200	310	2.82×10^7
	hls4ml (Ngadiuba et al.) [23]	95.0	260092	165513	0	345	200	190	4.94×10^7
JSC CERNBox	**KAN-Quantized-Pruned**	73.3	**1302**	**612**	0	0	338	8.9	$\mathbf{1.15 \times 10^4}$
	NeuraLUT-Assemble [4]	75.0	8539	1332	0	0	352	**5.7**	4.87×10^4
	AmigoLUT-NeuraLUT [34]	74.4	42742	4717	0	0	**520**	9.6	4.10×10^5
	PolyLUT-Add [20]	75.0	36484	1209	0	0	315	16	5.84×10^5
	NeuraLUT [3]	75.0	92357	4885	0	0	368	14	1.29×10^6
	PolyLUT [2]	**75.1**	246071	12384	0	0	203	25	6.15×10^6
	LogicNets [29]	72.0	37931	810	0	0	427	13	4.93×10^5
JSC OpenML	**KAN-Quantized-Pruned**	74.8	**1235**	623	0	0	315	9.5	1.17×10^4
	NeuraLUT-Assemble [4]	76.0	1780	540	0	0	**941**	**2.1**	$\mathbf{3.92 \times 10^3}$
	TreeLUT [16]	75.6	2234	**347**	0	0	735	2.7	6.03×10^3
	DWN [5]	**76.3**	6302	4128	0	0	695	14.4	9.07×10^4
	hls4ml (Fahim et al.) [9]	76.2	63251	4394	38	0	200	45	2.85×10^6

The edge of tomorrow

- EoT and hardware codesign
  - enable fundamentally new capabilities & robust adaptive systems
  - data challenge: connecting data across full-stack codesign from algorithms to devices to materials
- Fast and Slow together
  - Powerful autonomous instruments enabled through physics-inspired surrogate models and robust, real-time controllers

Feel free to join our Slack or mailing list, just reach out to me at ntran@fnal.gov