AMD

Brevitas & FINN

Building Inference Engines for Quantized Neural Networks

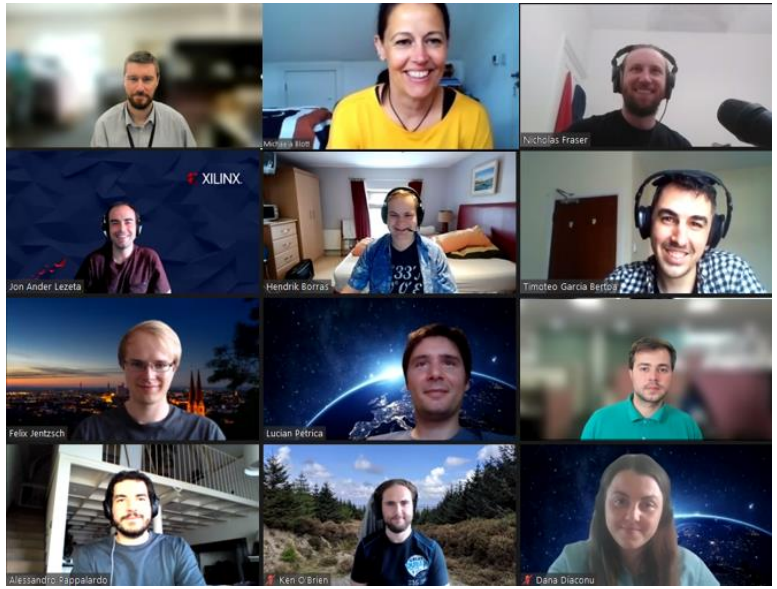
Thomas B. Preußer, AMD
4 July 2022

AMD AECG Research Group

- Established over 15 years ago (previously Xilinx Research)
  - Slowly expanding and increasingly leveraging external funding (IDA, H2020)
  - ~8 researchers + interns + university program

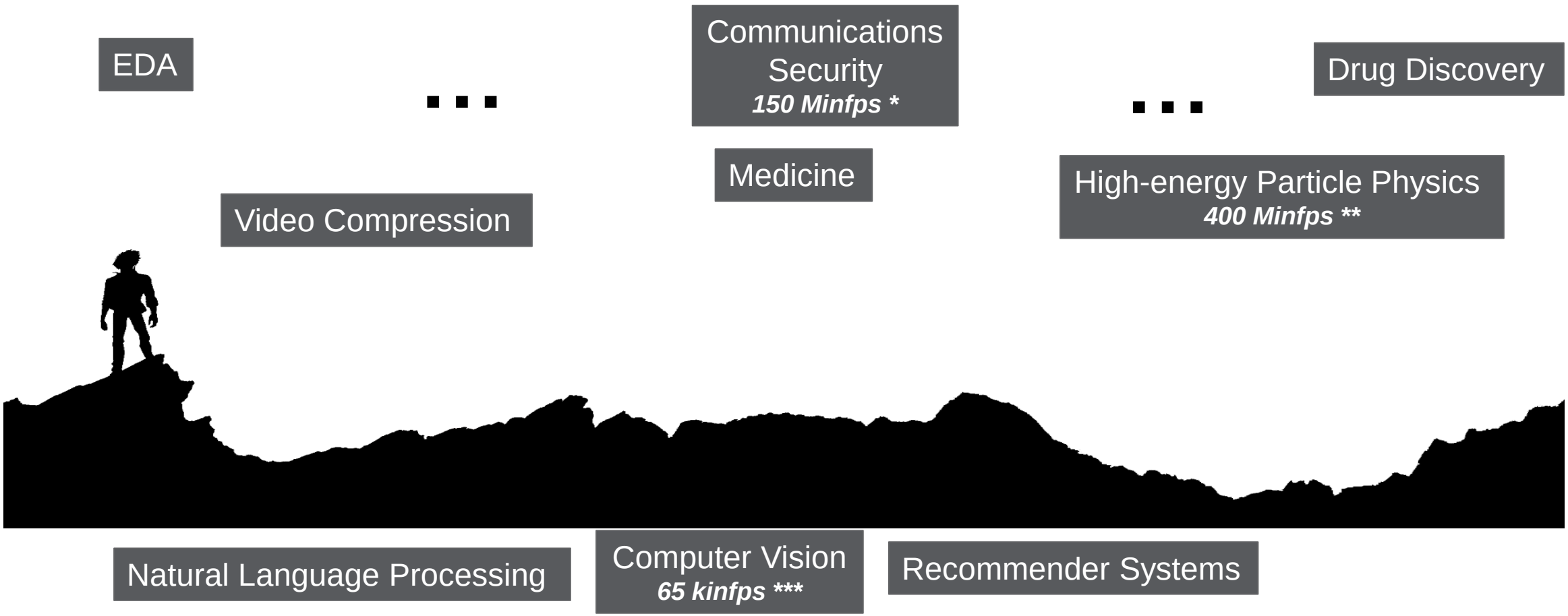
- Application-Driven Technology Development
  - Building systems, architectural exploration, algorithmic optimizations, benchmarking
  - Quantifying the value of our devices in novel applications
  - **AI and HPC**

- Close collaborations with customers, startups and universities
  - Automotive, security
  - ETH over 10 years, also: Strathclyde, Barcelona, Paderborn, and many others



Yaman Umuroglu, Michaela Blott, Thomas Preusser, Alessandro Pappalardo, Lucian Petrica, Nick Fraser, Jakoba Petri-Koenig, Ken O'Brien

Many Applications for Deep Learning



infps = inferences per second
**Network Intrusion Detection @100G*
***L1 trigger at CERN for CMS experiment (7 Tbps)*
****MLPerf 2020 Nvidia A100, ResNet50*

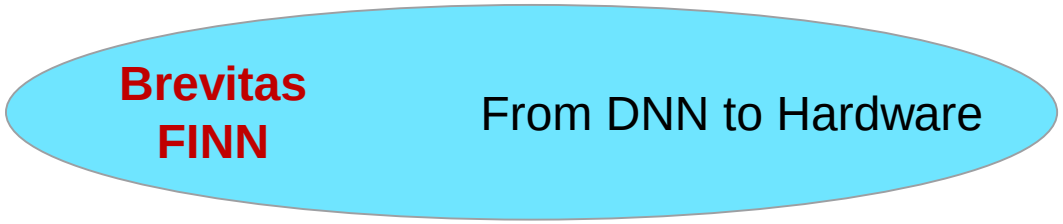
Many new ML Applications with very different requirements

AMD

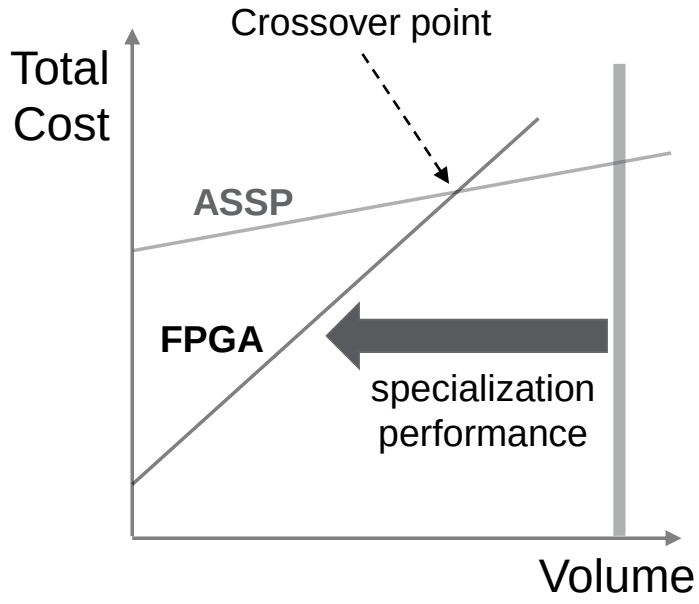
Motivation

- **ML is penetrating challenging embedded application domains:**
  video processing, computer vision, signal processing, communications, security
- **ML faces different challenging sets of requirements:**
  low real-time latency, high throughput, low power, small footprint
- **Architectural hardware specialization is key for success:**
  - Reduced-precision (`int8`) GeMM engines: Google TPU, Vitis AI
  - Spatially unrolled architecture: FINN
    - *Horizontal: Layer-specific* low-precision compute (1-4 bits), streaming dataflow
    - Vertical: Fine-grained sparsity exploitation

- **Need tools for:**
  - Quantization
  - Architecture compilation



Value Proposition: FPGA vs. Dedicated Silicon



- Specialization
  - buys performance but
  - loses sales volume.

FPGAs enable *affordable* custom specialization:

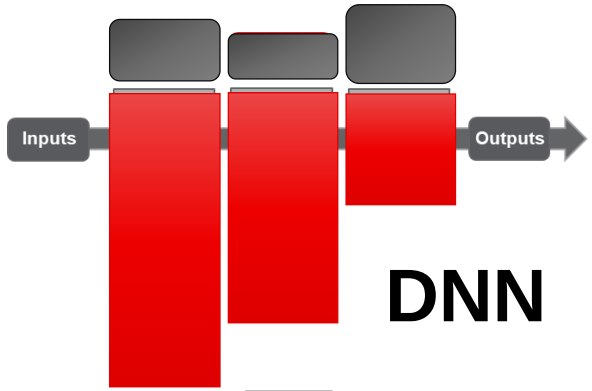
- Custom numeric precision
- Custom arithmetic
- Custom parallelism
- Custom …

We offer tools and solutions for a wide spectrum of customization needs.

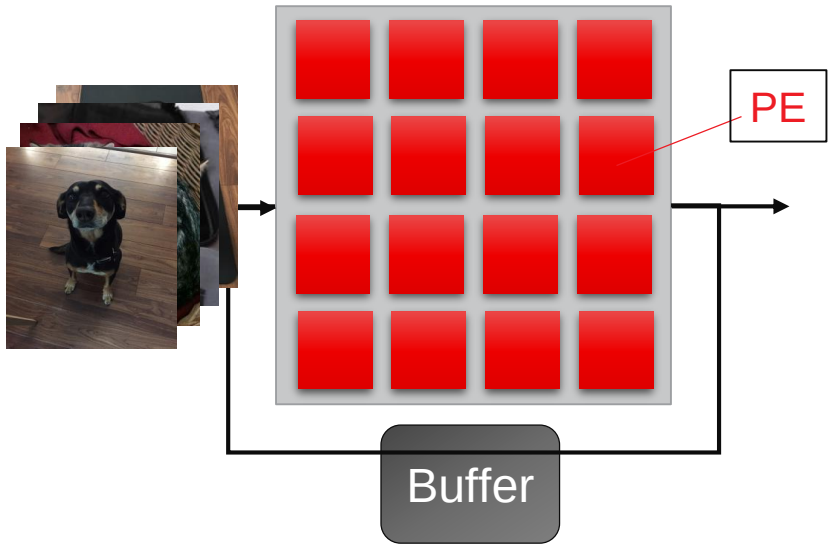
AMD

Industry's #1 Approach: Specialized Reduced-Precision GeMM Hardware

- Popular layer-by-layer compute
- Specialized processing engines
  - Operators
  - ALU types
    - tensor-, matrix- or vector-based
- Designed to run any DNN
- Gets you up to 10s kinfps
- Popular approach: VitisAI, Google's TPU

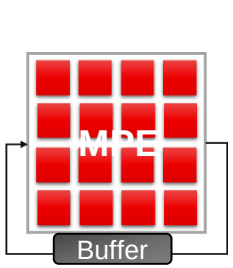


Matrix of Processing Engines (MPE)
"layer-by-layer" compute



AMD

Specialization beyond MPEs: Spatial Unrolling



Custom Dataflow

Quantization

Sparsity

Customized for ML in general

Customized for Specific Topologies

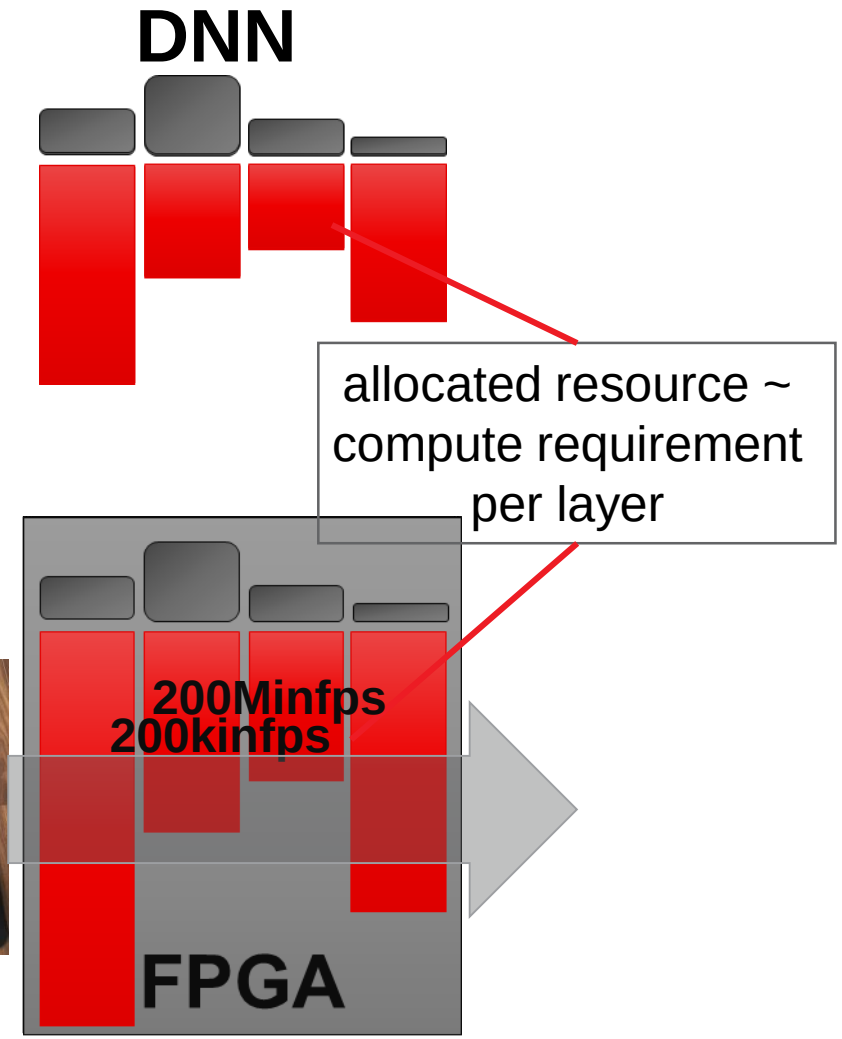
Customized in datatypes

Customized in connectivity

AMD

Dataflow - Specializing for Individual Topologies

- Hardware instantiates the topology as a dataflow architecture.
- Customize everything to the specifics of the given DNN, any operation, any connectivity.
- Benefits:
  - Improved efficiency, and
  - Low fixed latency.
- Scale performance & resources to meet the application requirements:
  - If resources allow, we can completely unfold to create a circuit that inferences at clock speed.

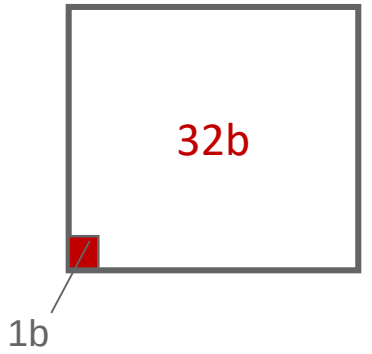


Dataflow can scale performance to meet the application requirements.

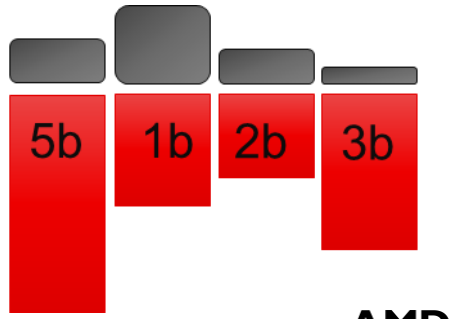
AMD

Customizing Arithmetic to Minimum Precision

- Popular approach reducing the bit widths for representing weights and activations while preserving accuracy.
- Reducing precision shrinks hardware cost, which may be invested into further performance scaling.
- Reduces memory footprint:
  - NN model can stay on-chip => no memory bottlenecks
- With dataflow: every layer has dedicated compute resources, we can mix and match precision across layers:
  - Exploit custom arithmetic at a greater degree than MPEs.



Precision	Modelsize [MB] (ResNet50)
32b	102.5
8b	25.5
1b	3.2

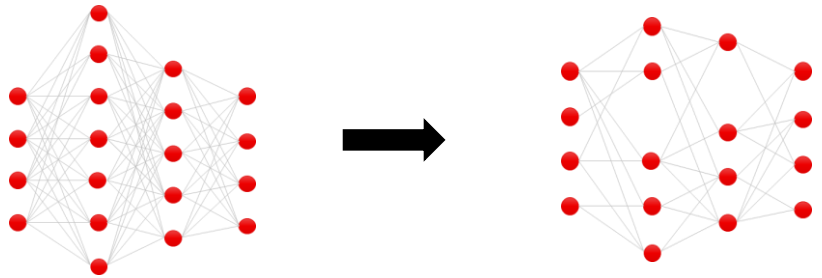


Reducing precision saves resources / scales performance, and reduces memory.
However, it requires quantization support in the training software.

AMD

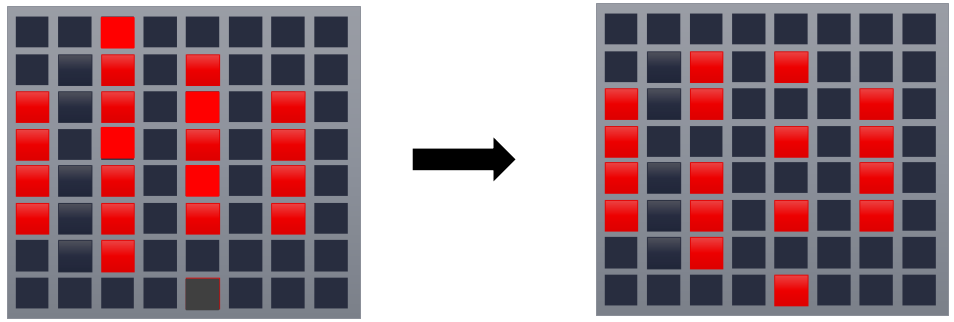
Sparsity

- DNNs are naturally sparse



- Sparse topologies result in irregular compute patterns, which are difficult to accelerate on vector- or matrix-based execution units.

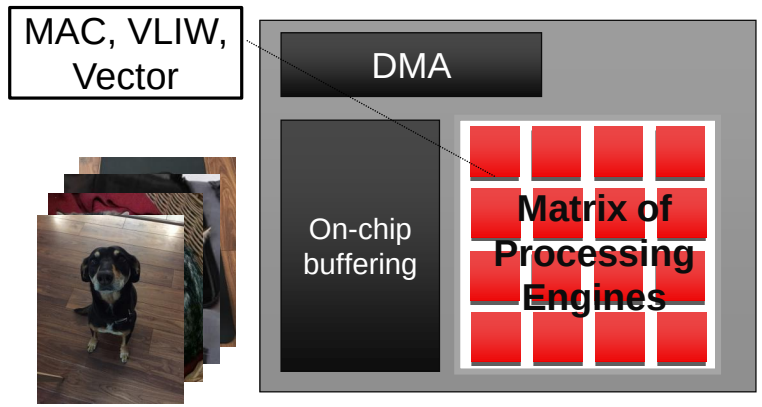
- With streaming dataflow architectures, where every neuron and synapse is represented in the hardware, we can fully exploit this.

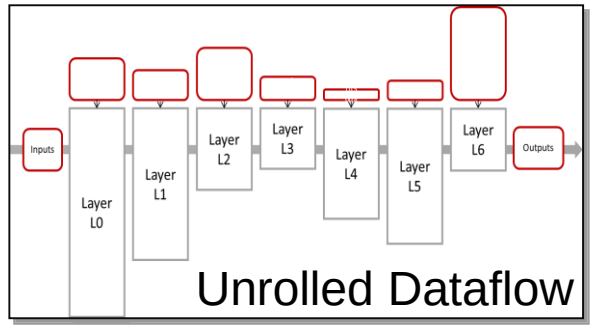


FPGA

Optimized Dataflow on FPGA

Levers for ML for Embedded and Edge Solutions





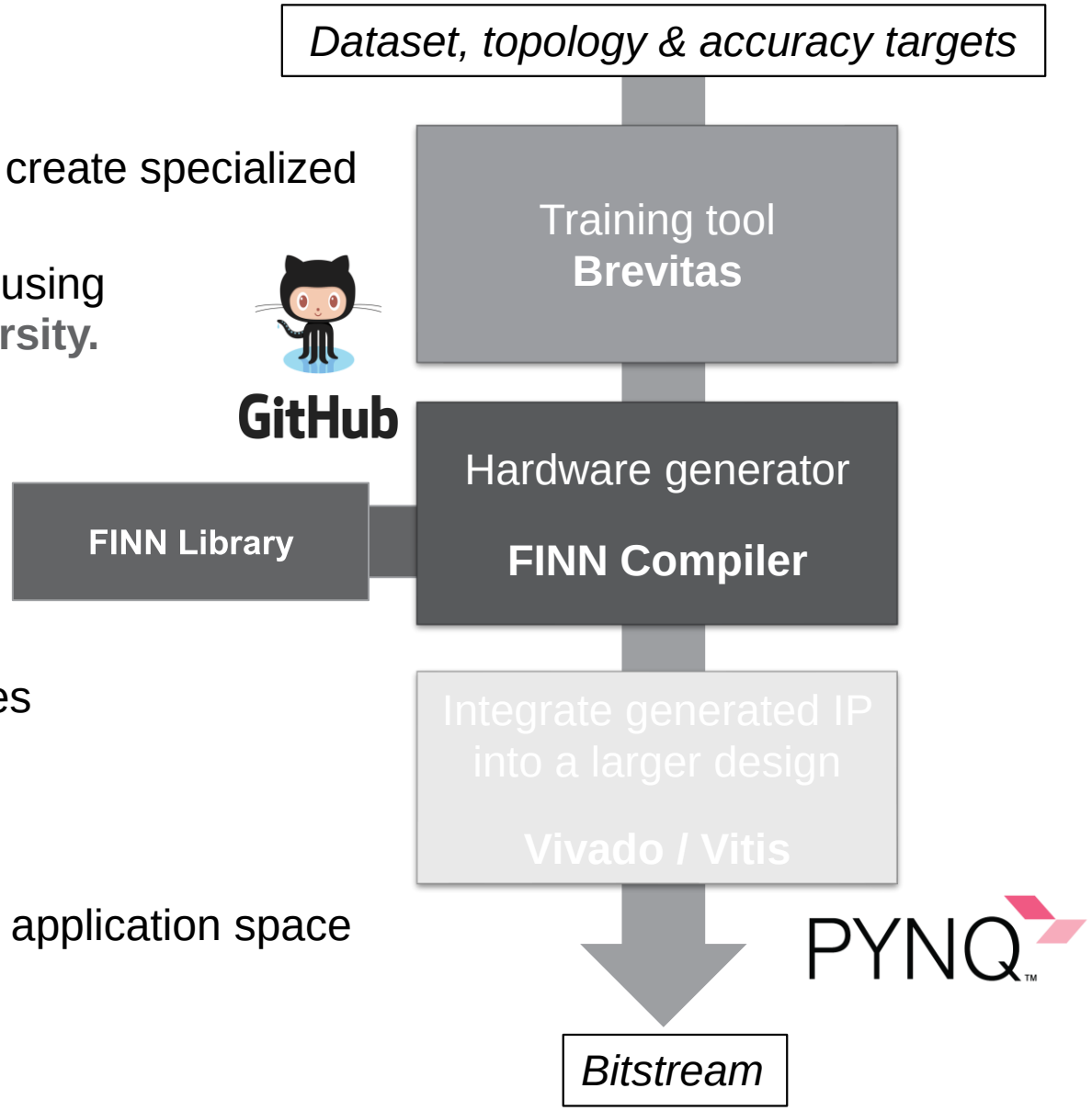
Batching	+ Bulk I/O + Locality: Parameter Caching + Higher Throughput - Increased Latency	**++**	**o**
Quantization	+ Reduced Storage + Reduced I/O + Simpler Arithmetic + Reduced Power - Retraining for Accuracy (particularly below 8 bits)	**+** safe compromise: typically 8 bits	**++** completely custom down to the layer
Sparsity	+ Reduced Compute - Irregular Access	**(+)** regular or much sparsity	**++** fine-granular sparsity (full unroll)
Solution		**Vitis AI: AI Engine, DSP**	**FINN, LogicNets: Fabric**

Project Mission

- End-to-end flow – from DNN to bitstream
  - Non-FPGA experts can train & quantize DNNs and create specialized hardware architectures on an FPGA.
  - Enables highly customized hardware architectures using **quantization** and **dataflow** and **fine-granular sparsity.**

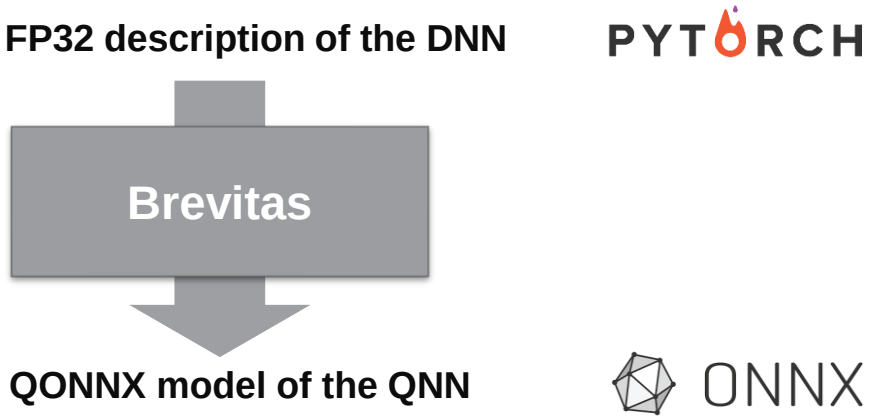
- Components
  - Brevitas: Quantization-aware training tool
  - FINN: Hardware generator
  - FINN Library: Parametrizable HLS/RTL HW modules

- Open source
  - Transparency and flexibility to adapt to fast-moving application space
  - Easy collaboration with customers

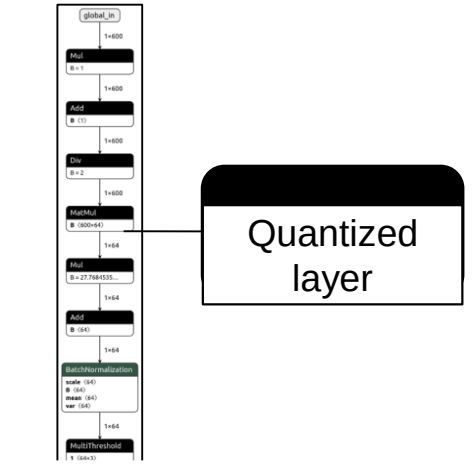


12

Brevitas:
A PyTorch Library for Training Quantized DNNs




```
model = nn.Sequential(
  nn.Linear(185, 100),
  nn.ReLU(),
  nn.Linear(100, 100),
  nn.ReLU(),
  nn.Linear(100, 100),
  nn.ReLU(),
  nn.Linear(100, 100),
  nn.ReLU(),
  nn.Linear(100, 1),
  nn.Sigmoid()
)
```



- Provides an interface familiar to ML engineers used to PyTorch
  - Floating point DNNs
- Quantizes to integer arithmetic
  - Supports different datatypes and can customize to match hardware arithmetic
- Exports to a commonly used representation format (QONNX)
  - Can be ingested by FINN and many other backends (e.g. FlexML)

The FINN Compiler

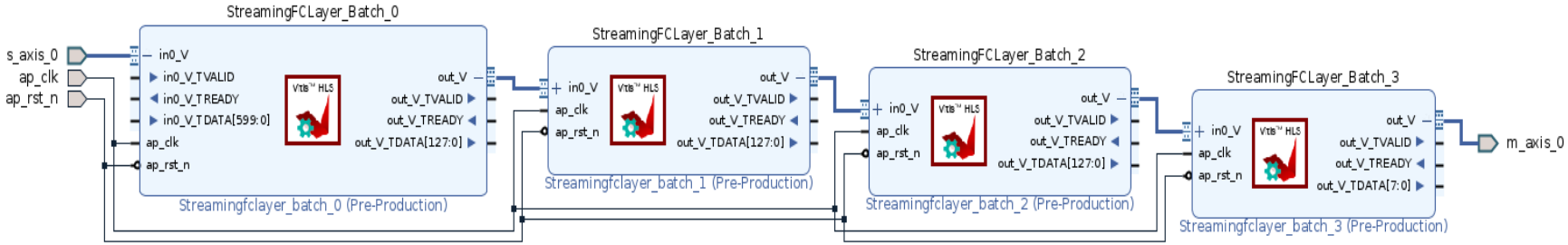
QONNX model of the QNN

Hardware Generation

FINN Compiler

Vivado IP

```
build.DataflowBuildConfig(
    # target performance and clock frequency
    target_fps           = 100 000 000,
    synth_clk_period_ns = 5.0,
    # target FPGA part number (e.g. for ZCU104)
    fpga_part            = "xczu7ev-ffvc1156-2-e",
    # ...
)
```

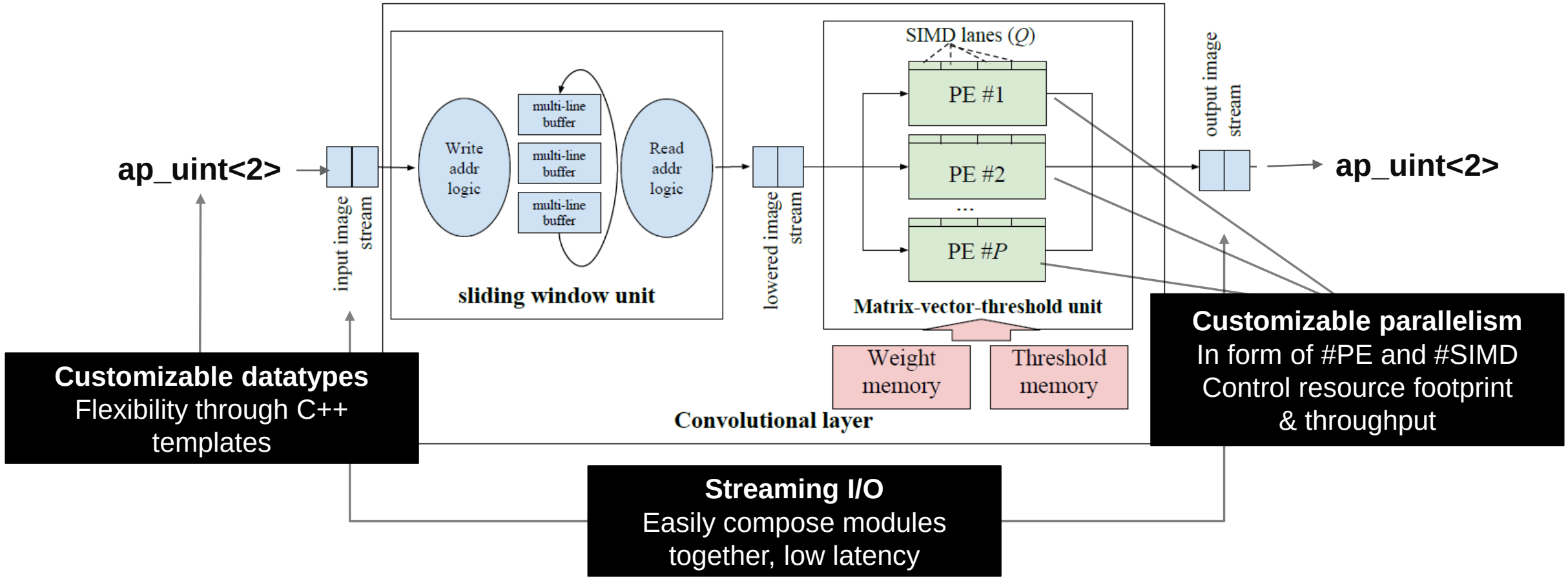


- Performs optimizations, operating on ONNX graph
- Generates code instantiating the layer implementations in a dataflow pipeline dimensioned to application requirements.
- Creates DNN hardware IP with AXI stream interfaces.

14

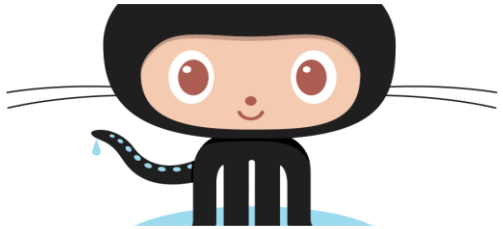
The FINN HLS Library

- An optimized, templated Vivado HLS C++ library of 10+ common DNN layers
- Key components: **Sliding Window Unit** & **MVTU (Matrix Vector Threshold Unit)**



Status

- **Open Source Adoption**
  - ~1.2k GitHub stars summarized across repos
  - 210k+ Brevitas downloads
  - 17k+ FINN compiler downloads



https://github.com/Xilinx/brevitas
https://github.com/Xilinx/finn
https://github.com/Xilinx/finn-hlslib

- **Academic results**
  - ACM TRETS 2020, FPL'2020, DFT'2019 Best Paper awards
  - 700+ citations on original paper

- **Universities building up computer architecture for ML classes with FINN**
  - Stanford, UNC Charlotte, NTNU in Norway, EPFL in Switzerland
  - Regular tutorials, also available on youtube: https://www.youtube.com/watch?v=zw2aG4PhzmA

- **Working with business units to scale-out support**

Hands-on Tutorial at FPL 2022 on 02-Sep-2022.


AMD