

Next Generation Triggers 1st Technical Workshop
25-27 November, 2024



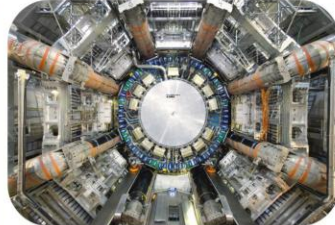
T1.2: Fast inference of complex network architectures on LHC online systems

[Dimitrios Danopoulos](#), Vladimir Loncar, Sebastian Dittmeier, Maurizio Pierini, Michael Kagan, Lorenzo Moneta, Roope Niemi, Chang Sun

Activities in Next Generation Triggers and T1.2



WP1: Infrastructure, Algorithms and Theory



WP2: Enhancing the ATLAS Trigger and Data Acquisition



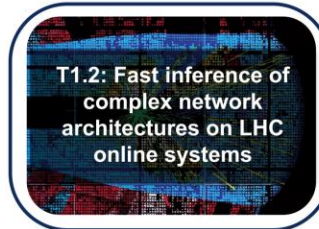
WP3: Rethinking the CMS Real-Time Data Processing



WP4: Education Programmes and Outreach



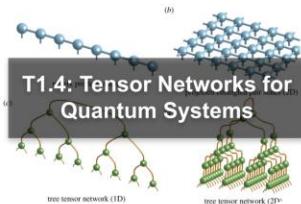
T1.1: Hardware and services for large scale NN optimisation and training, and physics simulation



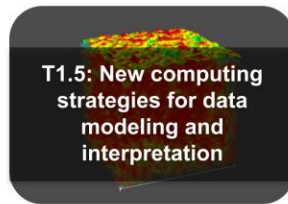
T1.2: Fast inference of complex network architectures on LHC online systems



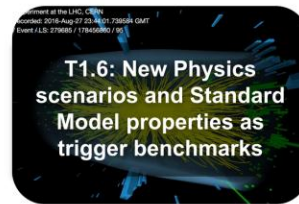
T1.3: Hardware-aware AI optimization



T1.4: Tensor Networks for Quantum Systems



T1.5: New computing strategies for data modeling and interpretation



T1.6: New Physics scenarios and Standard Model properties as trigger benchmarks



T1.7: Common software developments for heterogeneous architectures

Background and Motivation

Trigger systems are mechanisms that quickly decide which collision events to filter for further analysis

↳ ML has the potential to improve current algorithms

The HL-LHC is expected to produce **significantly** higher collision rates.

Challenges with High-Luminosity LHC:

- Handling the increased data
- **Need for speed and precision:** Efficient, high-speed ML processing is crucial to identify rare physics events without losing valuable data.

What's needed? ➡ new Infrastructure, Algorithms and Theory.

Fast Inference of Complex Network Architectures

T1.2 Goal: Develop hardware-efficient neural networks and tools for FPGA deployment, while promoting collaborative efforts.

Next-Gen Inference Engine

Develop a hardware-aware inference engine and an advanced code-generation core.



Utilize and expand [hls4ml](#): source-to-source compiler creating HW-optimal HLS designs for FPGAs ([docs](#) | [git](#))

Community Impact

Ensure the work benefits the experiment-specific work packages (i.e., [WP2](#) and [WP3](#)), but also the FastML/hls4ml communities.

T1.2 Deliverables and Timelines

Time	Description	Deliverable/Milestone
6 m	Demonstrator of Knowledge Distillation workflow to real-life LHC use cases	Integration in hls4ml on multiple backends
12 m	- Deployment of transformers on FPGAs - Demonstrator of Knowledge Distillation workflow to real-life LHC use cases	- Integration in hls4ml on multiple backends - Journal publication on Knowledge Distillation on Transformer use case
18 m	Support for generic Graph Neural Networks	- Improved code-generation infrastructure to support general graphs on multiple hls4ml backends - Journal publication on Graph NN fast inference
24 m	- Support for generic Transformer network - Mid-point hls4ml release	- Journal publication describing novel hls4ml functionalities and example applications - Tutorial describing new hls4ml functionalities
36 m	ASIC-oriented development and support for novel AI-specific hardware	- Prototype on specific AI hardware (to be identified) - Journal publication - Hosting the FastML workshop at CERN
48 m	Extended integration of common operators for AI engine	- Demonstrator on specific AI hardware (to be identified) - Journal publication
60 m	- Multi-FPGA support, for inference and optionally for training - Final hls4ml release	- Tutorial describing new hls4ml functionalities - Final hls4ml release - Journal publication

Internal milestones are flexible, guided by community feedback.
Roadmaps for specific features are created based on this input.

Commitment to the yearly workshops and software releases.

Year	Code	Milestones	Type
1	M1.1.2	MLonFPGA community workshop	event/report
2	M2.1.2	hls4ml software release 1	software
4	M4.1.2	hls4ml & NNLO software release 2	software
5	M5.1.2	hls4ml & NNLO software release 3	software

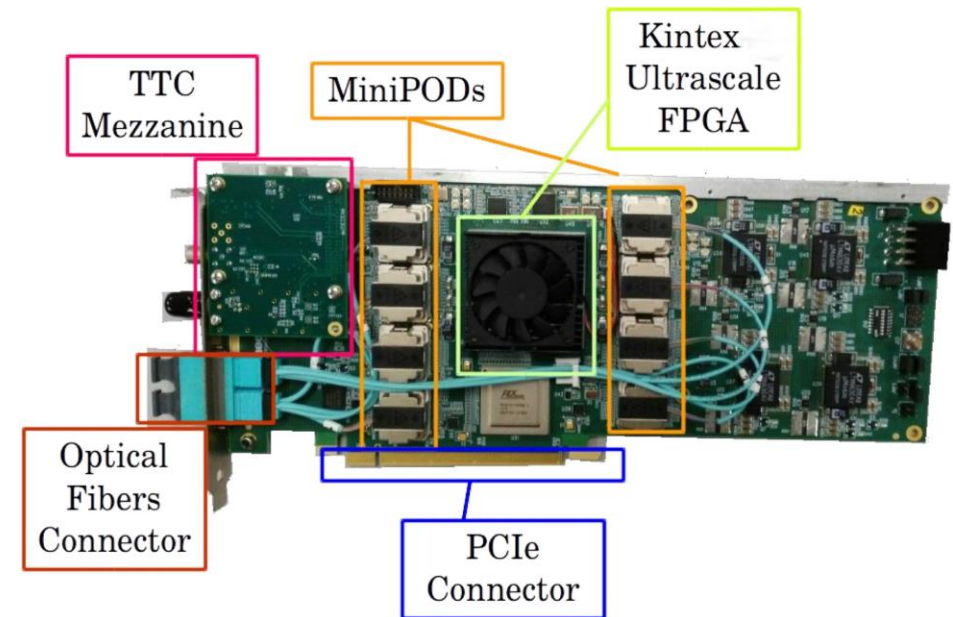
Key feedback from the community

- Strong interest for integration of **Xilinx AI engines**
- Interest in GNNs, DeepSets, SNNs, and more “exotic” architectures
- Mixture of low latencies and accelerator card designs
- Easy setup for hyperparameter optimizations (related to T1.3)
- hls4ml build server
- Enhanced reconfigurability (weights, architecture)
- Model registry

Real-Time Processing with FPGAs

Why Use FPGAs at LHC:

- They can achieve **ultra-low latency** in deterministic time.
- They have customizable architecture, allowing **ML-specific** configurations.



FPGA used in the ATLAS Detector [[link](#)]

Programming FPGAs with High Level Synthesis

Traditional FPGA Programming

- Used languages like VHDL and Verilog
- Required in-depth knowledge of digital design principles
- Time-consuming design/verification process

Transition to High-Level Synthesis (HLS):

- HW abstraction via a high-level language (C/C++)
 - ↳ use directives to guide the compiler to optimize the HW design
- HLS tools automatically generate RTL code from higher-level code.
- Significantly reduces development time and complexity.

```
#include "mv.h"

void mv(
    unsigned int A[MAX_SIZE*MAX_SIZE],
    unsigned int b[MAX_SIZE],
    unsigned int c[MAX_SIZE]){

    #pragma HLS INTERFACE s_axilite port=A bundle=data
    #pragma HLS INTERFACE s_axilite port=b bundle=data
    #pragma HLS INTERFACE s_axilite port=c bundle=data

    for(int i = 0; i < ELEM; ++i){
        c[i] = 0;

        #pragma HLS PIPELINE II=1

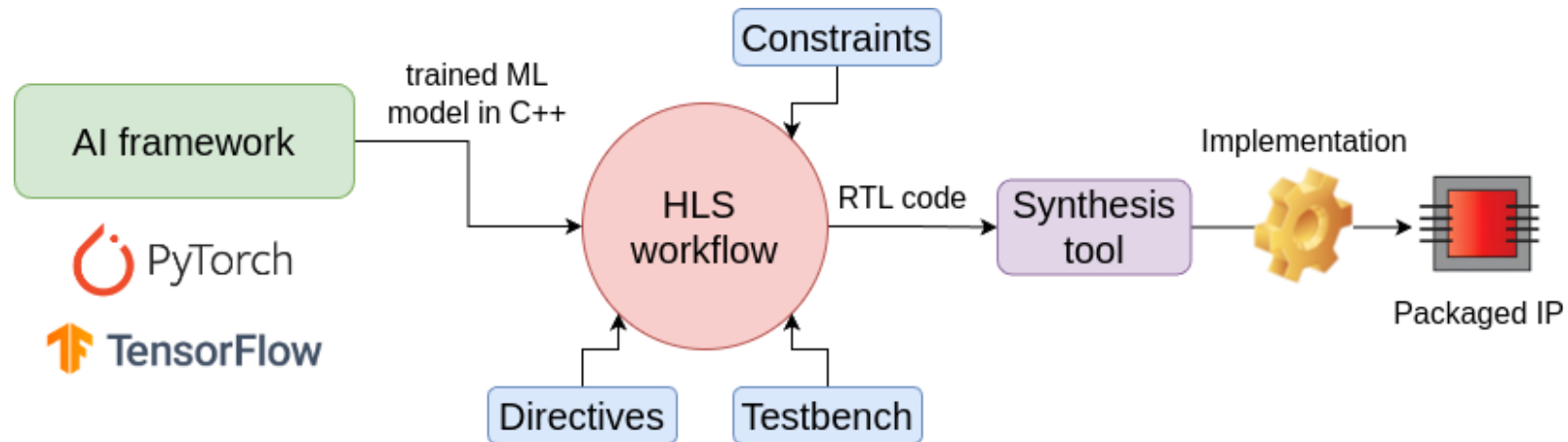
        for(int j = 0; j < ELEM; ++j){
            c[i] += A[i*ELEM + j]*b[j];
        }
    }

    return;
}
```

HLS example function

Designing ML models for FPGAs

Convert a trained ML model into a C++ representation for use in HLS workflow



Using HLS to design ML models for FPGAs remains cumbersome and time-consuming:

1. HLS firmware design requires expertise
2. Achieving optimal performance is hard: a lot of parameters and strategies to consider
3. Debugging and verification are challenging due to the long synthesis times, even for small ML models

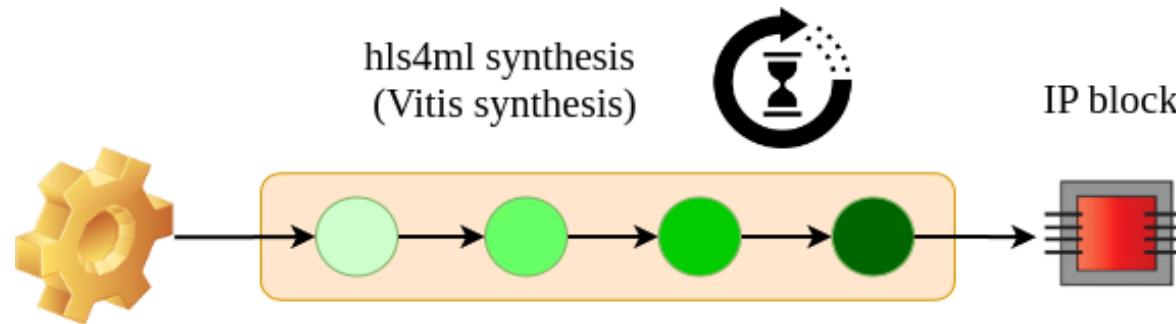
Utilizing hls4ml for fast FPGA inference

hls4ml: source-to-source compiler that creates HW-optimal HLS designs for FPGAs ([docs](#) | [git](#))

- Automatic translation of trained ML models to optimized HLS code.
 - ↳ Currently supporting common ML architectures - MLP, CNN, RNN
 - Ease of use through automation while providing flexibility for customization.
-
- Main focus of T1.2 throughout the 5-year project
 - Widely explored at CERN for online processing (HW triggers)
 - Enables rapid deployment and acceleration of ML models with ultra-low latency
 - Large community of users @ FastML

Initial focus: Reducing hardware synthesis time

Problem: The synthesis of complex ML models as a whole using hls4ml can be lengthy (often >24h)



Solution: Partition the model graph in hls4ml into smaller, independent subgraphs.

↳ allows each subgraph to be synthesized individually in parallel.

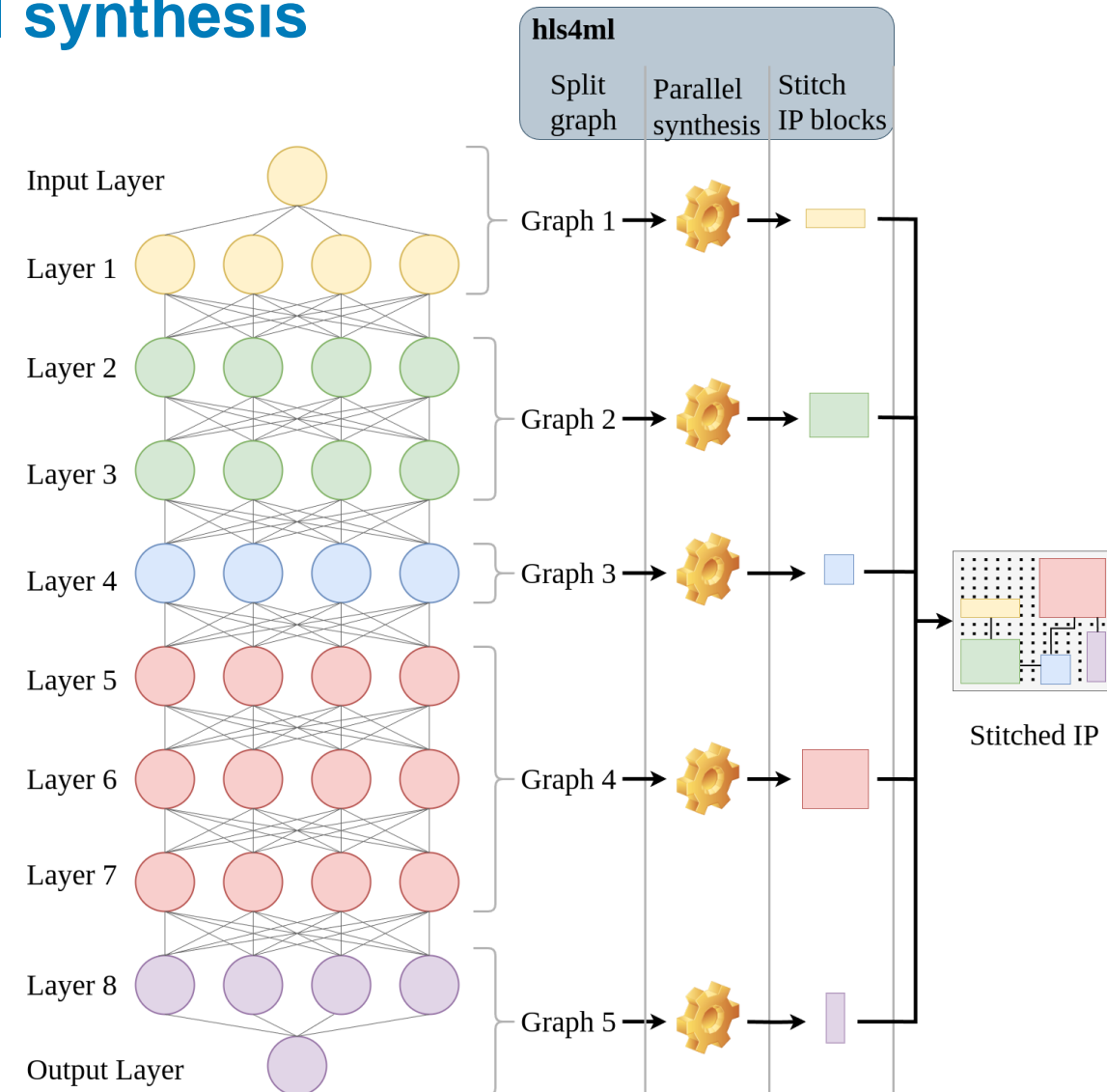
Splitting the AI model graph for parallel synthesis

Workflow:

- **Select layers as splitting points**
 - currently chosen by the user
 - future plans for being automated by hls4ml
- **Parallel synthesis**
 - Independently & concurrently synthesize each subgraph
 - Export each subgraph as an independent IP block.
- **IP block stitching**
 - automatic merging in Vivado to create the final IP

Synthesis
status

graph1:		graph2:		graph3:	
graph1:		graph2:		graph3:	
graph1:		graph2:		graph3:	
graph1:		graph2:		graph3:	
graph1:		graph2:		graph3:	



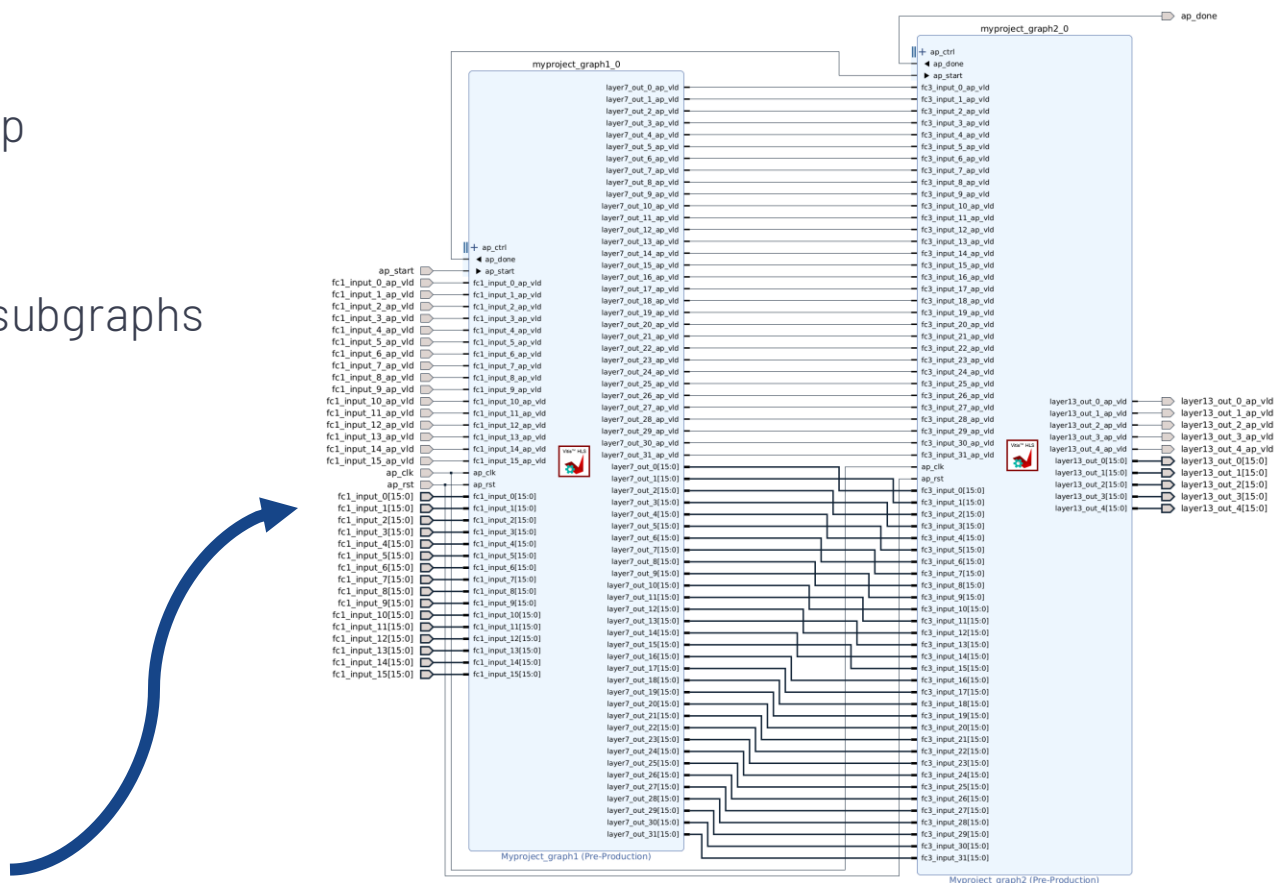
Results

- Support for partitioned and streaming IP interfaces
- ~3x decrease in synthesis time based on current setup
- Greater reductions are expected for larger models.

Theoretically, the decrease ratio can match the No. of subgraphs

MLP example

Model	DSP	FF	LUT	Latency
Original	2870	23719	122284	15 cycles
Graph 1	2071	16966	89807	5 cycles
Graph 2	799	6799	34177	9 cycles
Stitched	2870	23765	123984	14 cycles



Automatic IP stitching in Vivado

Conclusion on graph partitioning

Benefits at a glance:

- **Time efficiency:** Significantly reduces synthesis and build times.
- **Scalability:** Enables efficient handling of larger models by partitioning.
- **Flexibility:** Simplifies debugging of subgraphs without requiring full model resynthesis.
- **Latency reduction:** Potential improvements observed in some models during synthesis.
- **AI framework-agnostic:** the partitioning is handled internally by hls4ml.

T1.2 future plans

Short term:

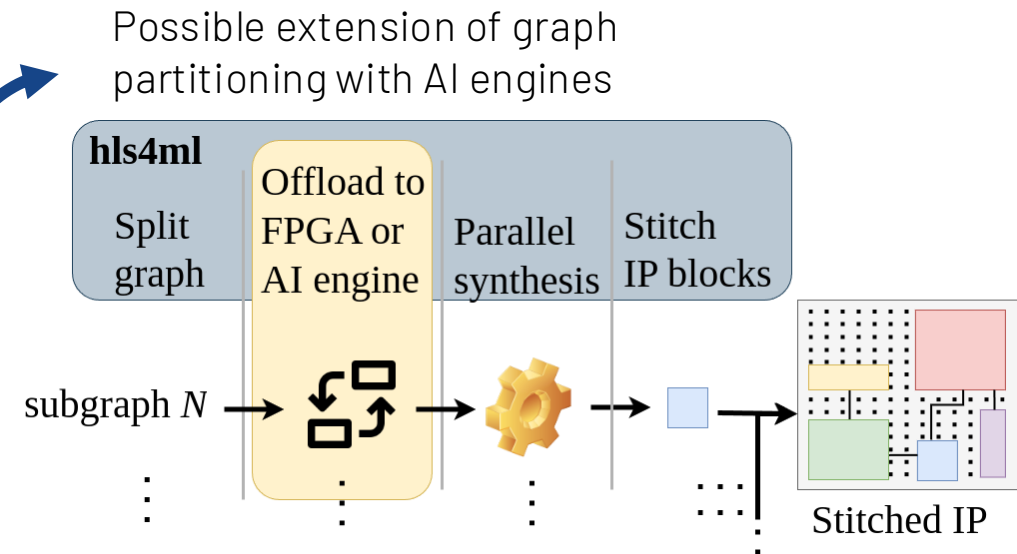
- Refine the graph splitting and IP stitching processes
 - ➡ Add support for larger and more complex models (i.e., models with skip connections)
 - ➡ Add support for simulation of the stitched IP
 - ➡ Offer a seamless workflow for the user

Medium term:

- Investigate interesting ML model architectures
 - ➡ compact transformers, graph NNs, etc.
- Exploration of AI engines based on community feedback

Long term:

- Develop an advanced code generation core
- Explore Tree Tensor Networks (in collaboration with T1.4)



Thank you!

