AI and real-time systems with Aidge

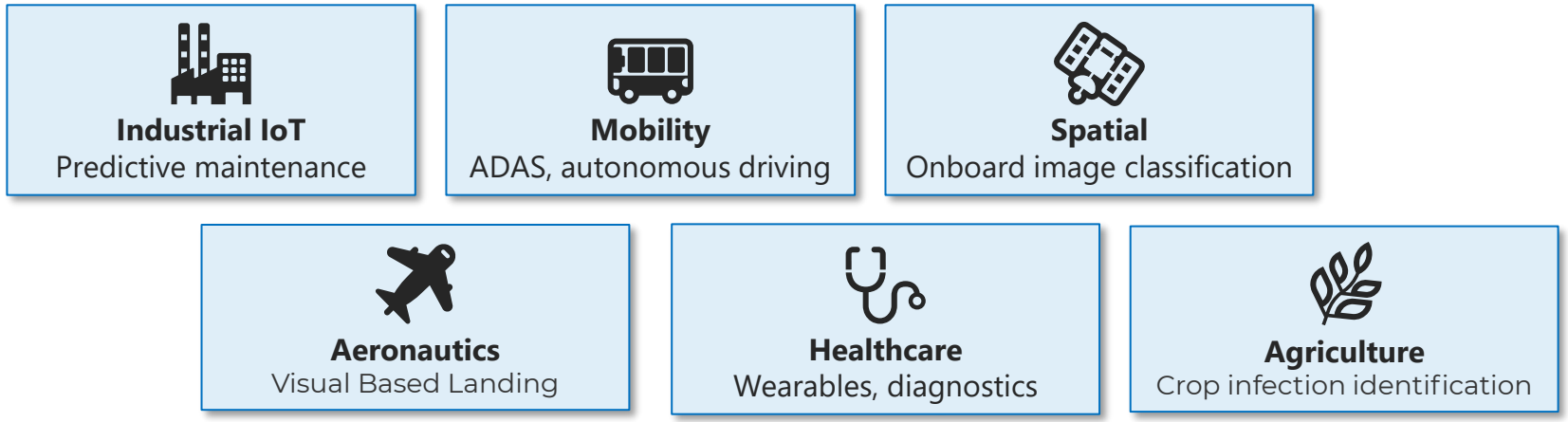
Iryna de Albuquerque Silva - Research engineer @ Embedded AI Lab, CEA List

Tutorial @ ETR 2026 – Poitiers, France

September 10th, 2026

AI and real-time systems

Use cases



AI and real-time systems

Key challenges

- Predictability → AI models are optimized for *average-case latency*, not *worst-case bounds*
  - Dynamic inputs: input-dependent control flows (conditions) and variable-length sequence processing (Transformers, LLMs, RNNs);
  - Complex hardware: GPUs, TPUs, and modern NPUs rely on dynamic branch prediction, memory caching, and parallel thread warping;
- Reliability / safety → AI models are opaque non-linear learned functions
  - Out-of-Distribution (OOD) inputs: edge-cases are usually incorrectly treated with high confidence [1];
  - Expensive verification: traditional standards like DO-178C (avionics) and ISO 26262 (automotive) impose heavy certification process;

[1] Hendrycks & Gimpel, ICLR 2017

Structural limitations of ML toolchains

Deep Learning Frameworks



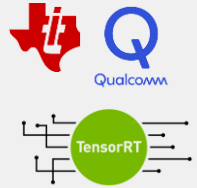
- Runtime dependencies
- Limited interoperability
- Closed-source execution

Optimization Compilers



- Low-level approach
- Extremely complex pipelines

Hardware-Specific SDKs



- Closed source, black-box
- Vendor lock-in restricting portability

Leading to

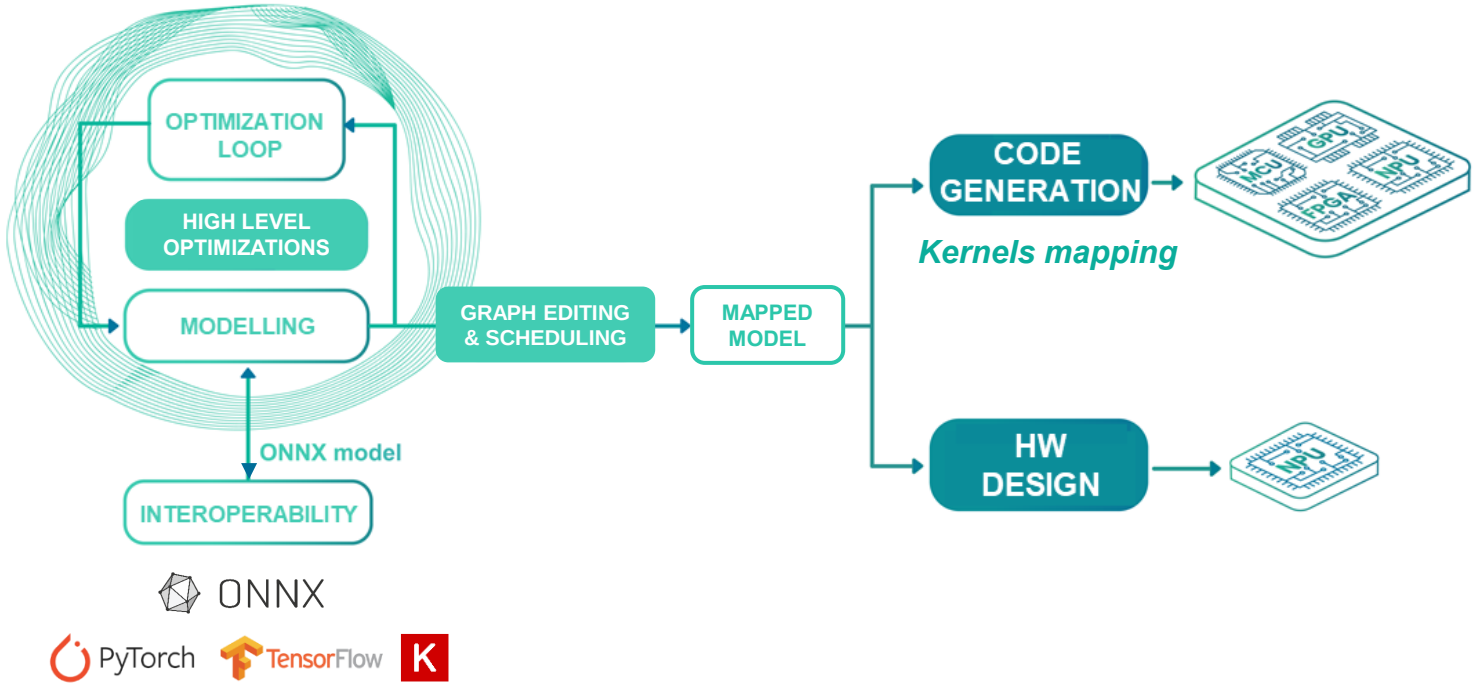
Fragmented solutions

Expertise scarcity

Innovation bottleneck

Time & cost multiplier

Aidge: a ML framework dedicated to embedded AI



End-to-end interoperable framework including optimization and deployment

Modular library with light core, plugins system and few dependencies

Open source and transparent code hosted by ECLIPSE FOUNDATION

API

OS and Packaging

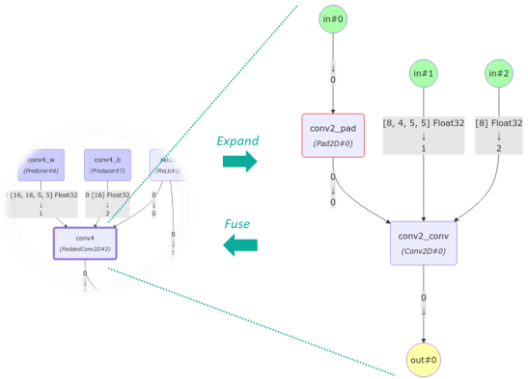
Windows Mac ubuntu PIP

A complete framework with extendable modules



Design Optimize Deploy

Efficient Intermediate Representation



Easily manipulate the computational graph

Seamless interoperability

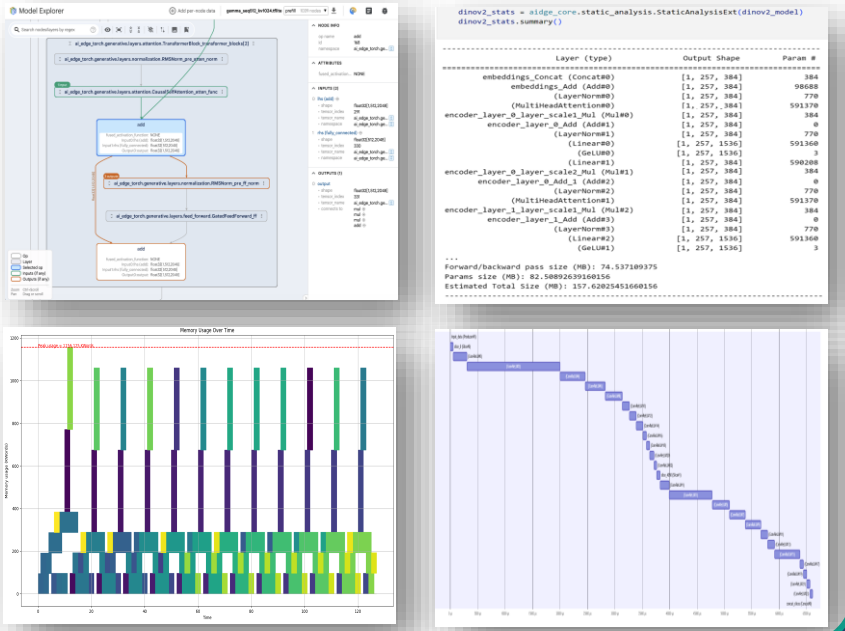
ONNX

PyTorch TensorFlow K

OpenCV NumPy

Support most popular **CNN, RNN and SNN architectures**

Tools for in-depth analysis

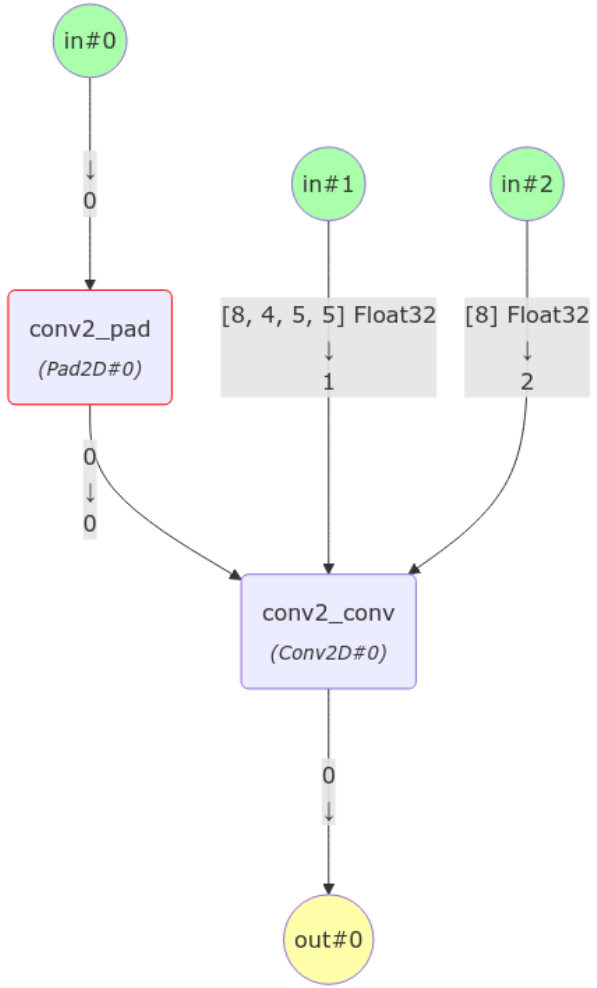


What sets Aidge apart?

- The right *Intermediate Representation (IR)* is key!
- Aidge translates high-level AI models into a robust and flexible IR perfectly equipped to support real-time constraints.

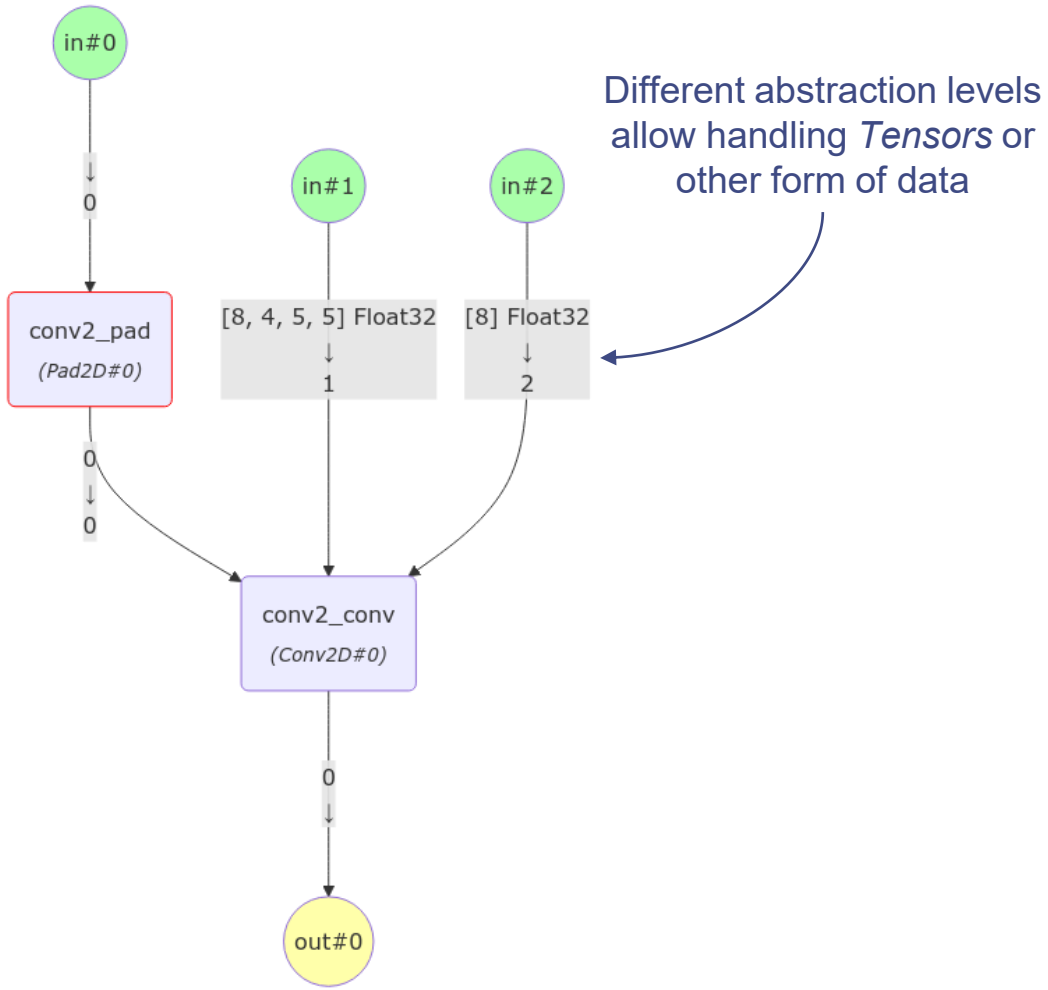
Core IR features

- Aidge translates high-level AI models into a robust and flexible IR perfectly equipped to support real-time constraints. It is:
  - **Transparent**



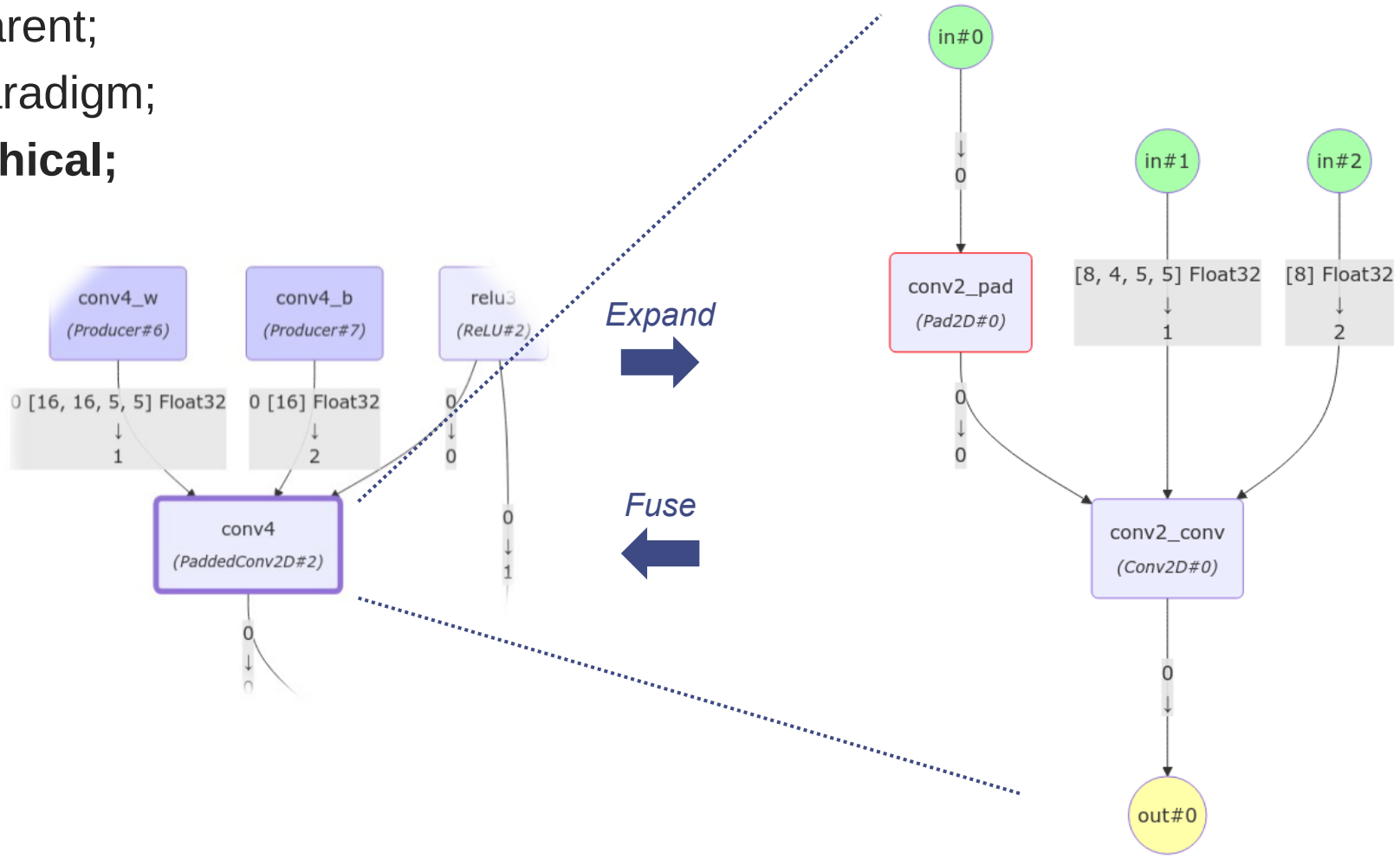
Core IR features

- Aidge translates high-level AI models into a robust and flexible IR perfectly equipped to support real-time constraints. It is:
  - Transparent;
  - **Multi-paradigm;**



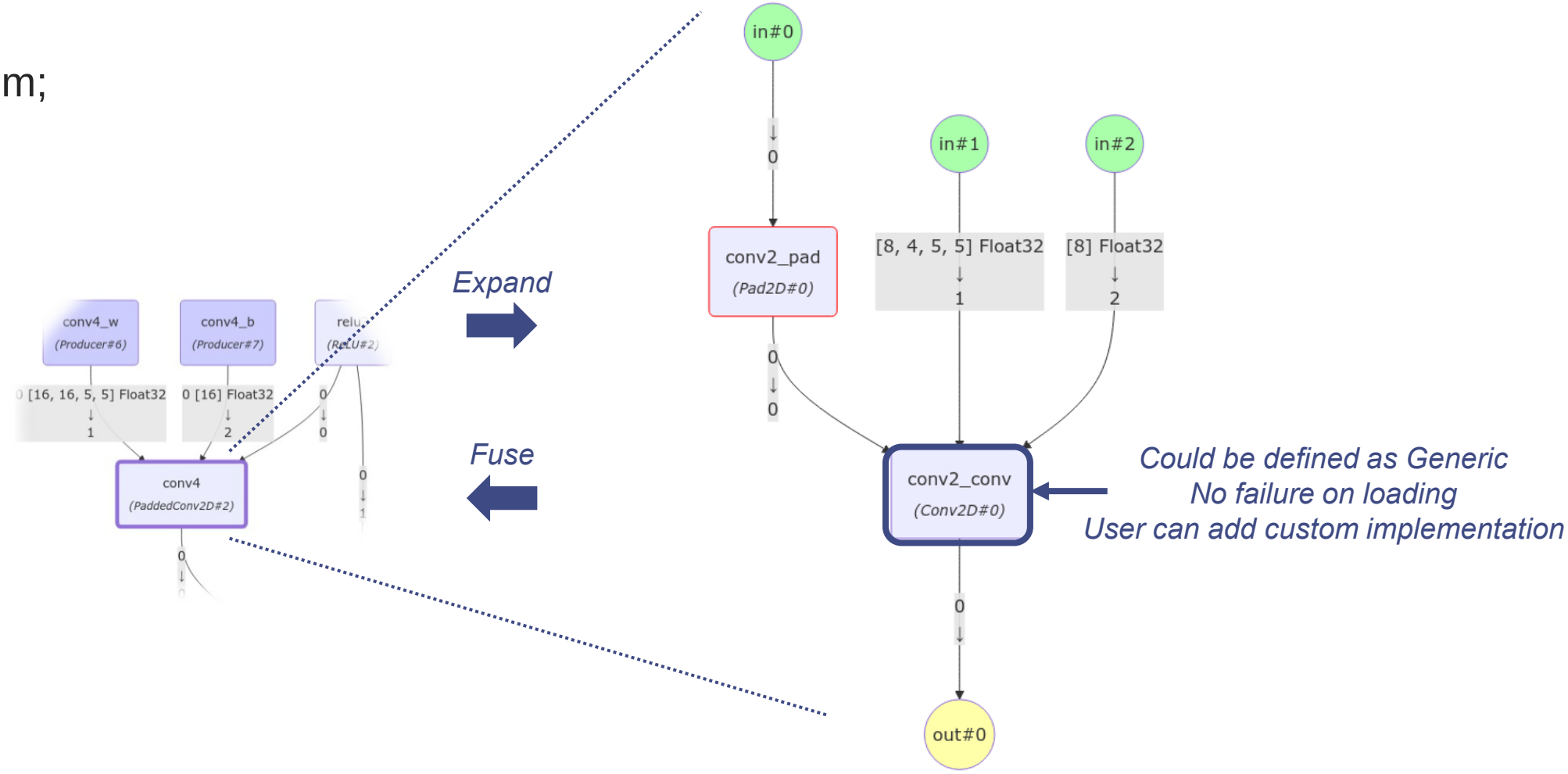
Core IR features

- Aidge translates high-level AI models into a robust and flexible IR perfectly equipped to support real-time constraints. It is:
  - Transparent;
  - Multi-paradigm;
  - **Hierarchical;**



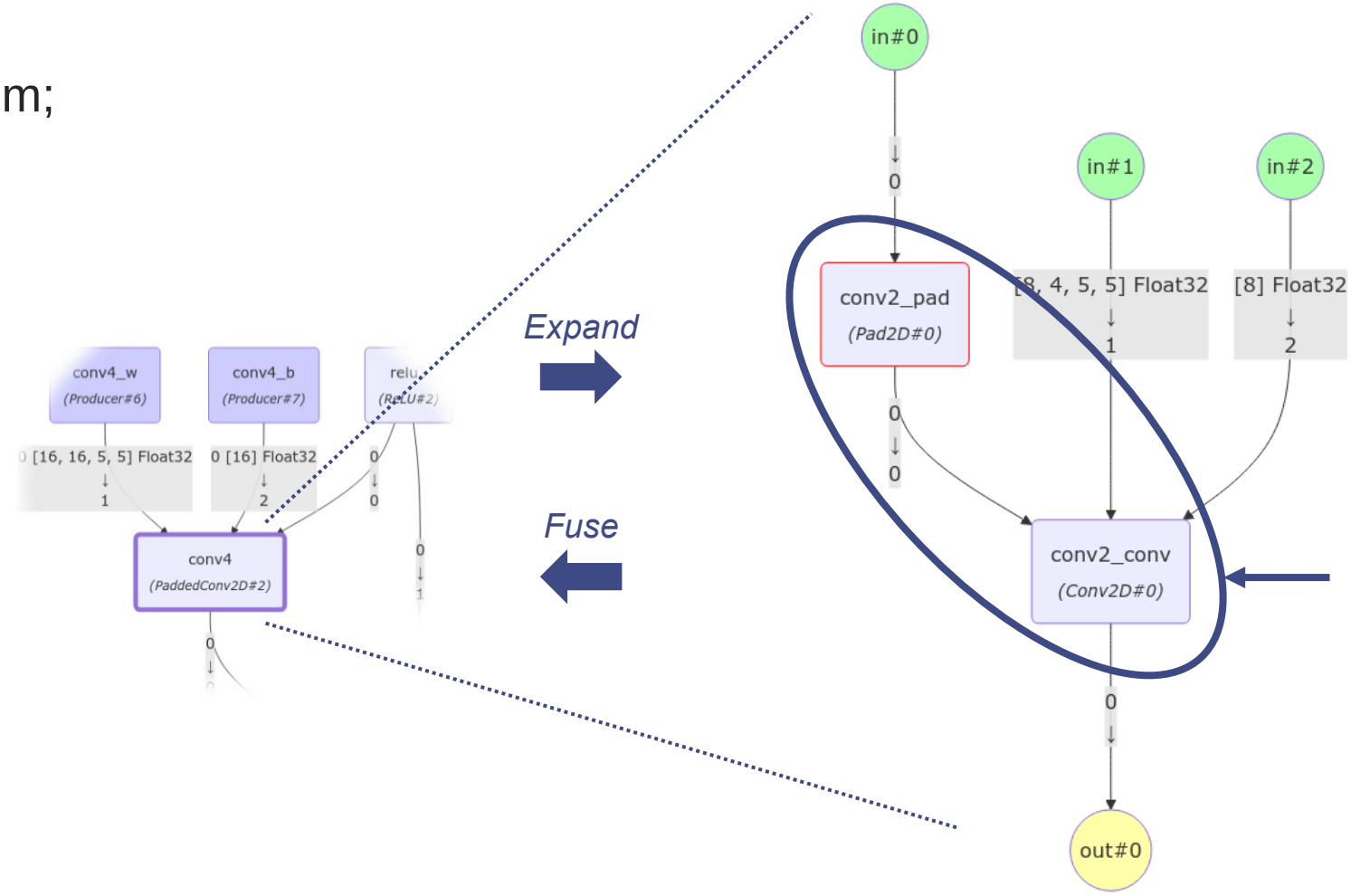
Core IR features

- Aidge translates high-level AI models into a robust and flexible IR perfectly equipped to support real-time constraints. It is:
  - Transparent;
  - Multi-paradigm;
  - Hiearchical;
  - **Generic**



Core IR features

- Aidge translates high-level AI models into a robust and flexible IR perfectly equipped to support real-time constraints. It is:
  - Transparent;
  - Multi-paradigm;
  - Hiearchical;
  - Generic;
  - **Searchable**



Graph RegEx allows to search for arbitrarily complex patterns

Front-end model integration

Seamlessly interface with widely adopted AI workflows:

- Two-way *ONNX (Open Neural Network Exchange)* format interoperability, with more than 100 operators supported;
- Direct *PyTorch* interface integration (on-going).

```
graph = aidge_onnx.load_onnx("model.onnx")

...

aidge_onnx.export(graph, "optimized_model.onnx")
```

ONNX ➔ Aidge

```
@aidge_onnx.node_import.auto_register_import("swish")
def import_swish(onnx_node, input_nodes, opset=None):
    node_name = onnx_node.output[0]
    return gen_swish_metaop(node_name)
```

Aidge ➔ ONNX

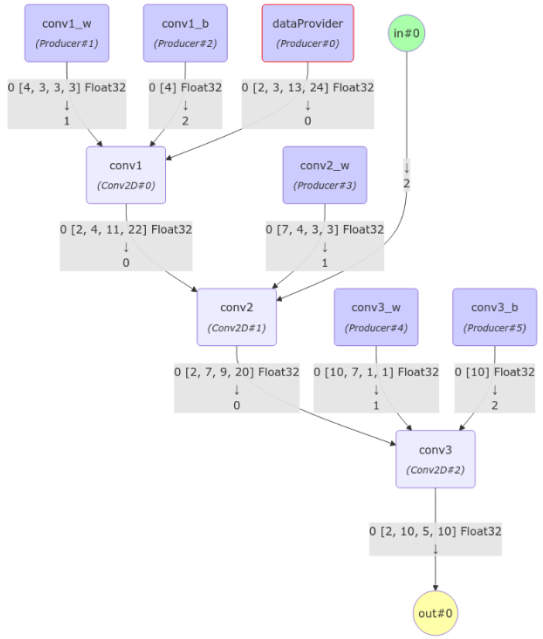
```
@aidge_onnx.node_export.auto_register_export("BitShift")
def export_bitshift(aidge_node: aidge_core.Node, node_inputs_name,
node_outputs_name, opset=None):
    aidge_operator = aidge_node.get_operator()
    onnx_node = helper.make_node(
        name=aidge_node.name(),
        op_type="BitShift",
        inputs=node_inputs_name,
        outputs=node_outputs_name,
    )
    onnx_node.attribute.append(helper.make_attribute("direction",
        DirectionAttrToString(aidge_operator.attr.get_attr("direction"))
    ))
    return [onnx_node]
```

Advanced AI graph manipulation

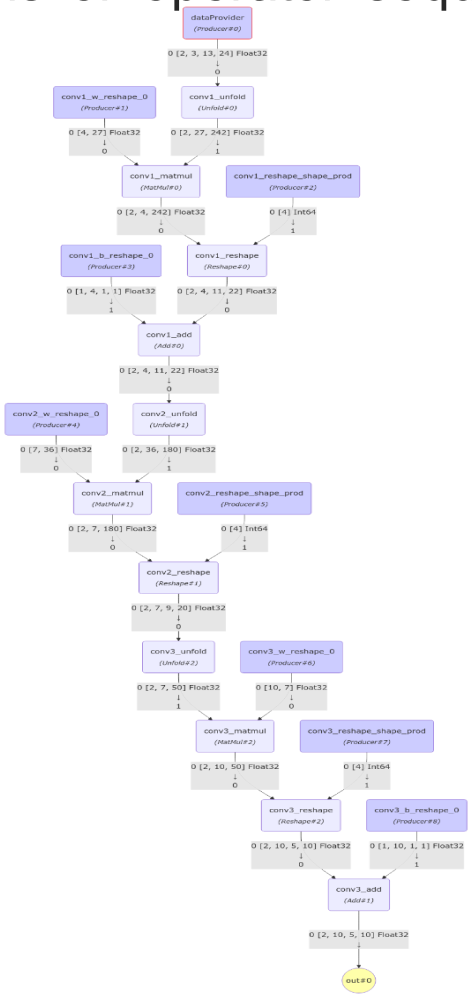
Tailor graph transformations to match specific layout requirements:

- **Patterns search and replacement:** locate specific subgraphs or operator sequences and replace them with optimized custom operators instantly;

```
aidge_core.conv_to_matmul(graph)
```



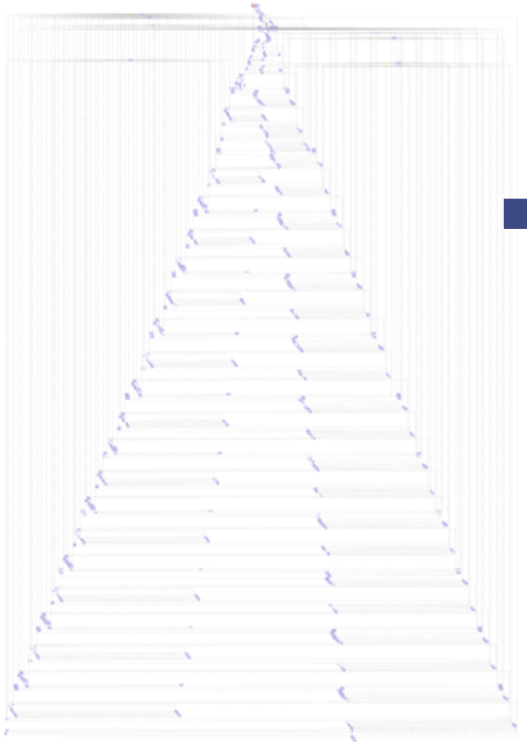
Replace Conv with Unfold + MatMul operators



Advanced AI graph manipulation

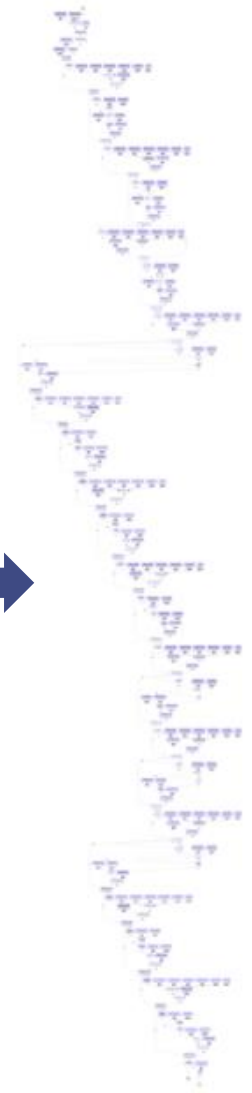
Tailor graph transformations to match specific layout requirements:

- **Adaptable graph granularity:** group arbitrary operators recursively into high-level "Meta-Operators" or lower complex blocks down to execution primitives;



Thousands nodes DinoV2 ONNX…

```
dinov2_model = aidge_onnx.load_onnx("dinov2.onnx")
aidge_core.fuse_to_metaops(dinov2_model, "MatMul-*>Add", "Linear")
aidge_core.fuse_to_metaops(dinov2_model, "ReduceMean-*>Sub#1~>(Pow#1->ReduceMean-*>Add#1->Sqrt);"
                                         "Sub#1~*>Div#1;"
                                         "Pow#1<1~Producer;"
                                         "Add#1<*~Producer;"
                                         "Mul#1<*~Producer;"
                                         "Add#2<*~Producer;"
aidge_core.fuse_to_metaops(dinov2_model, "MatMul->Div#1->Softmax-*>MatMul;"
                                         "Div#1<1~Producer", "ScaledDotProductAttention")
aidge_core.fuse_to_metaops(dinov2_model, "ScaledDotProductAttention#1->Transpose->Reshape#1->Linear;"
                                         "Reshape#1<1~Producer;"
                                         "ScaledDotProductAttention#1<0-(Transpose<-Reshape#2<-Add#1);"
                                         "ScaledDotProductAttention#1<1-(Transpose<-Reshape#3<-Add#2);"
                                         "ScaledDotProductAttention#1<2-(Transpose<-Reshape#4<-Add#3);"
                                         "Reshape#2<1~Producer;"
                                         "Add#1<*-0-Split#1;"
                                         "Add#2<*-1-Split#1;"
                                         "Add#3<*-2-Split#1;"
                                         "Split#1<-MatMul;"
                                         "Split#1<1~Producer", "MultiHeadAttention")
```

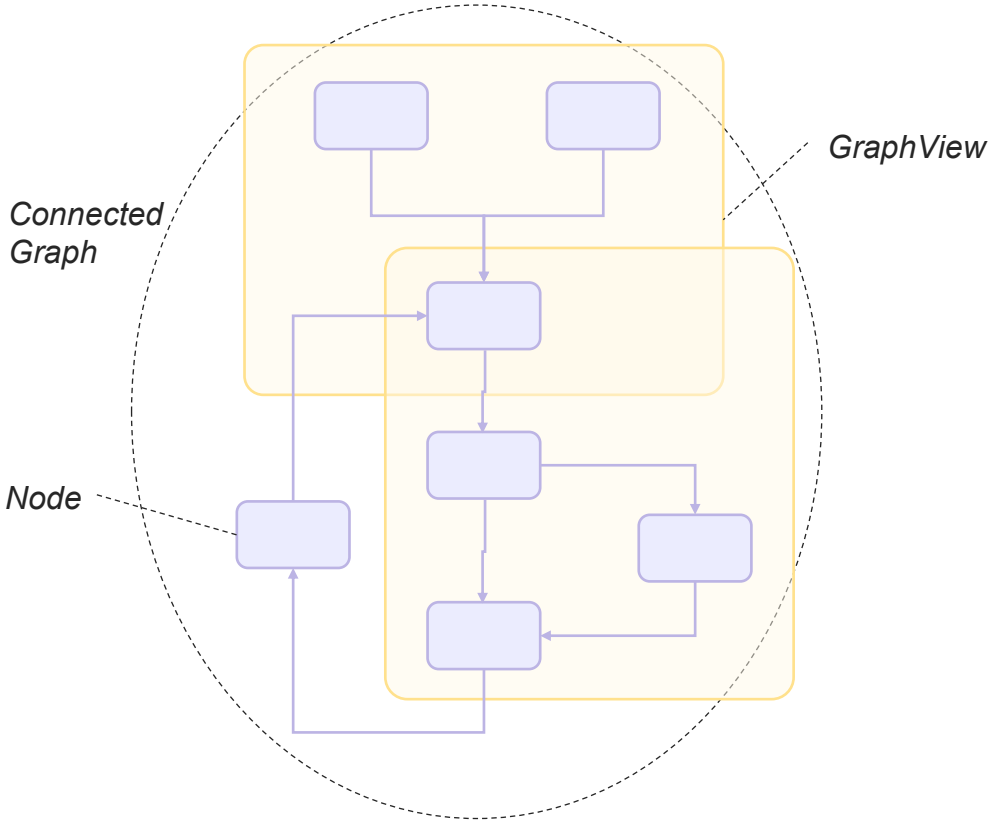


Graph at the required granularity for your use-case

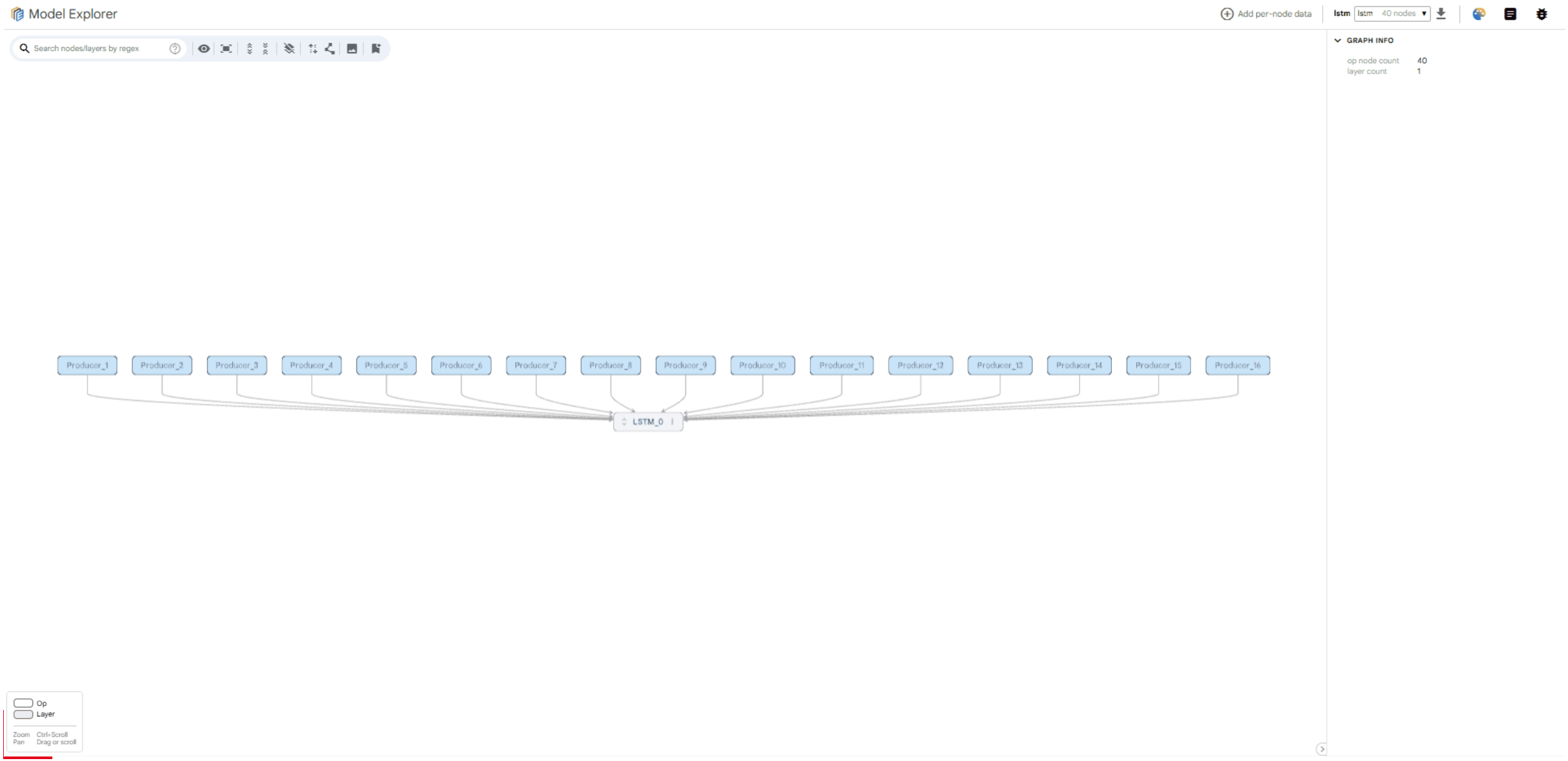
Advanced AI graph manipulation

Tailor graph transformations to match specific layout requirements:

- **Decoupled topology views**: create arbitrary subgraph views to manage scheduling, parallel loops, or data tiling pipelines without altering the semantics of the original graph.



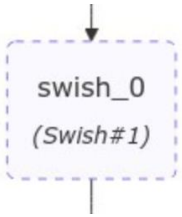
Graph visualizer



Novel operator definition

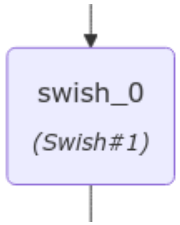
Deploy custom functional blocks with two distinct approaches:

- ***GenericOperator* abstraction for prototyping**: define a new hardware function or mathematical model completely on-the-fly via the Python API;
- **Native operator integration:** embed specialized operators directly into Aidge's C++/Python codebase, for native reusability and integration across the entire framework.




```
    # Create a generic op. of type Swish
    node_swish = aidge_core.GenericOperator("Swish",
[aidge_core.InputCategory.Data], 1)
    swish_op = node_swish.get_operator()
    # Optional: add attributes to the generic op.
    swish_op.attr.beta = [1.0]
    # Optional: specifies how its output dimensions are computed w.r.t. its inputs
    swish_op.set_forward_dims(lambda x: x)
```






```
class SwishImpl(aidge_core.OperatorImpl):
    def __init__(self, op: aidge_core.Operator):
        aidge_core.OperatorImpl.__init__(self, op, "swish")
        self.op = op
    def forward(self):
        data_input = np.array(self.op.get_input(0))
        beta = np.array(self.op.attr.get_attr("beta"))
        output = (data_input / (1 + np.exp(-data_input*beta)))
        self.op.set_output(0, aidge_core.Tensor(output))

swish_op.set_impl(SwishImpl(swish_op)) # Setting implementation
```

Define an implementation

Bind the implementation to the operator

Operator C-P model customization

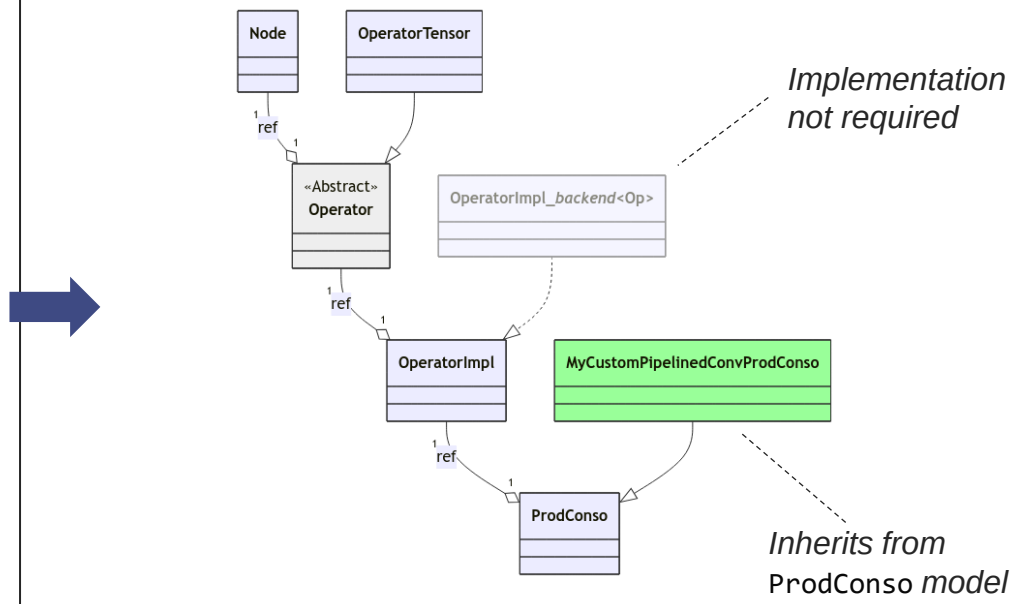
Leverage Aidge's well-defined consumer-producer (C-P) model:

- **Flexible data abstraction:** specify how much data is consumed and produced by the operator implementation at each execution step, using precise number of elements or tokens.
- **Decoupled execution paths**: isolate the C-P scheduling path from actual data processing to enable graph scheduling without concrete operator implementations.

```
class Conv2D_DataFlow_CP(aidge_core.ProdConso):
    def __init__(self, op: aidge_core.Operator):
        aidge_core.ProdConso.__init__(self, op)
        self.state_begin = True

    def get_nb_required_data(self, input_idx):
        input = self.get_operator().get_input(input_idx)
        if input:
            if self.state_begin:
                # Require 3 lines for 3x3 kernel
                return aidge_core.Elts_t.data_elts(3 * input.dims()[2])
            else:
                return aidge_core.Elts_t.data_elts(input.dims()[2])
        else:
            return aidge_core.Elts_t.none_elts()

    def get_required_memory(self, output_idx, inputs_size):
        output = self.get_operator().get_output(output_idx)
        self.state_begin = False
        return aidge_core.Elts_t.data_elts(output.dims()[2])
```



Execution scheduling model

- Aidge's scheduling is **static**
  - Get early and late logical static scheduling for any graph (cyclic or acyclic)

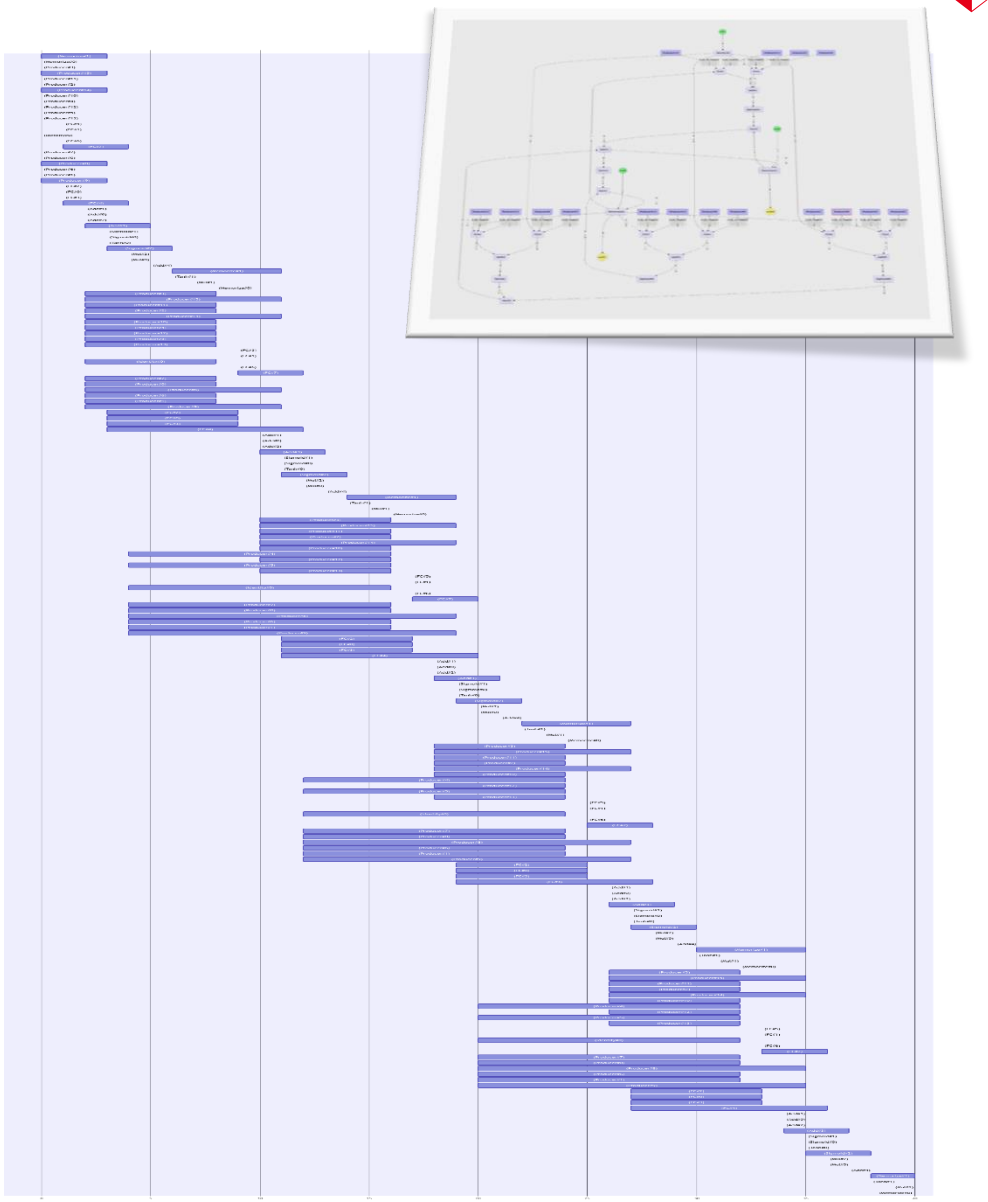
```
lstm = aidge_core.LSTM(in_channels=4, hidden_channels=8, seq_length=5)

# Flatten the graph:
lstm_model = aidge_core.get_connected_graph_view(lstm)
aidge_core.expand_metaops(lstm_model)
lstm_model.set_backend("cpu")

# Create the Scheduler
lstm_scheduler = aidge_core.SequentialScheduler(lstm_model)
lstm_scheduler.generate_scheduling()

# Display static scheduling
lstm_scheduler.save_static_scheduling_diagram("lstm_scheduling")
```

  - Based on the specified Consumer-Producer model



Structural and static analysis

- End-to-end or layer-wise model *profiling*

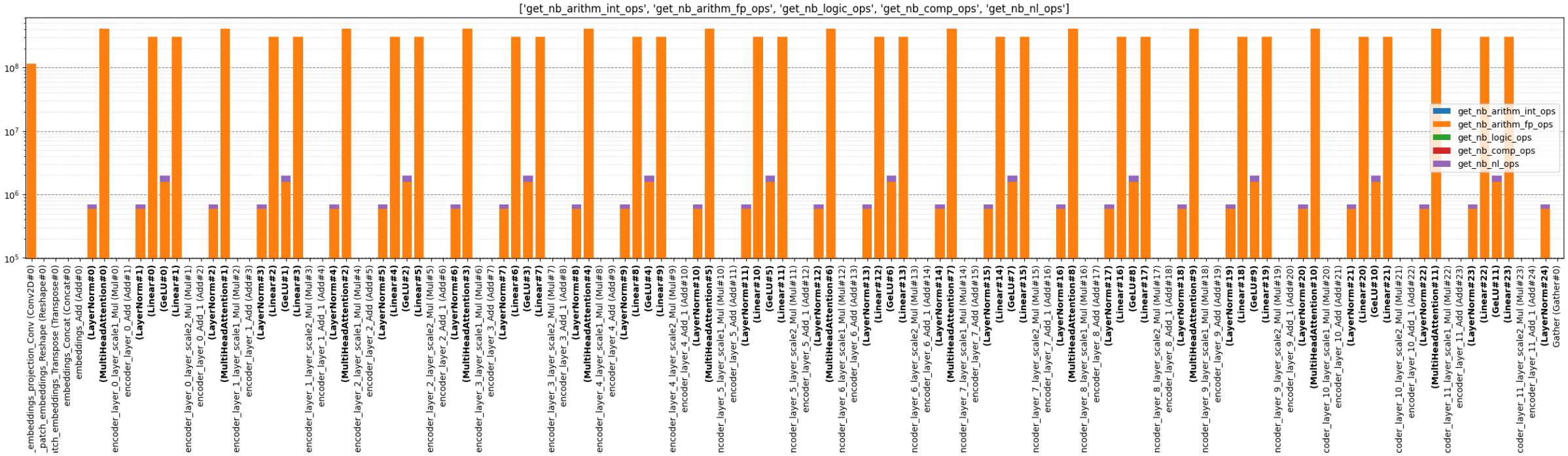
```
stats = aidge_core.static_analysis.StaticAnalysisExt(dinov2_model)
stats.summary() # Simple text output similar to Keras' model.summary()

stats.log_nb_ops_by_type("stats_ops.png", log_scale=True)
```

Rich low-level & high-level APIs:

- *Base C++: for lower-level heuristics integration*
- *Extended Python: with additional visualization methods*

Auto aggregation for meta-operators, specifiable for generic operators

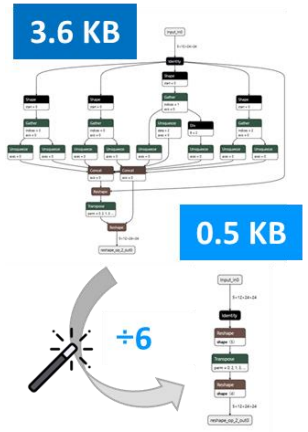


Design

Optimize

Deploy

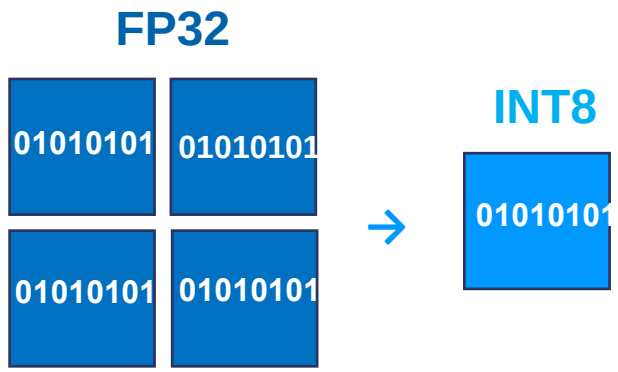
Model simplification



TensorRT Performance Comparison (1,000 iterations)

Model	Execution Time	Throughput	vs Best
original	0.552 ms	1812.0 inf/s	+0.007 ms (+1.3%)
onnx_cleaner	0.545 ms	1835.4 inf/s	BEST
onnxsim	0.550 ms	1817.1 inf/s	+0.005 ms (+1.0%)
onnxslim	0.554 ms	1806.1 inf/s	+0.009 ms (+1.6%)

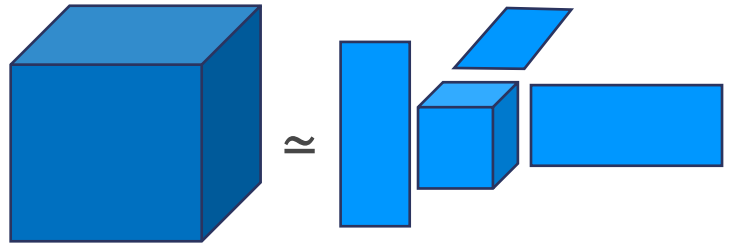
Quantization
Post or Aware Training



Choose the precision level

FP32 → INT8 → INT4 → INT2

Tensor compression



60% smaller and

30% faster

Simplification *recipes*

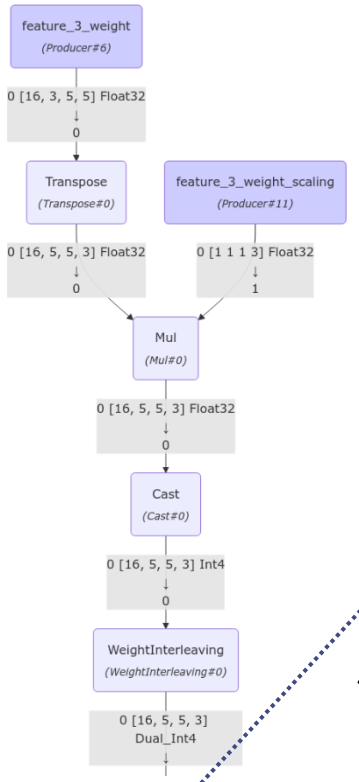
- Have a clean, step-by-step, graph transformation:
  - Policy: do not change the data, change the graph!

Adapt the memory layout

Apply the quantization scaling

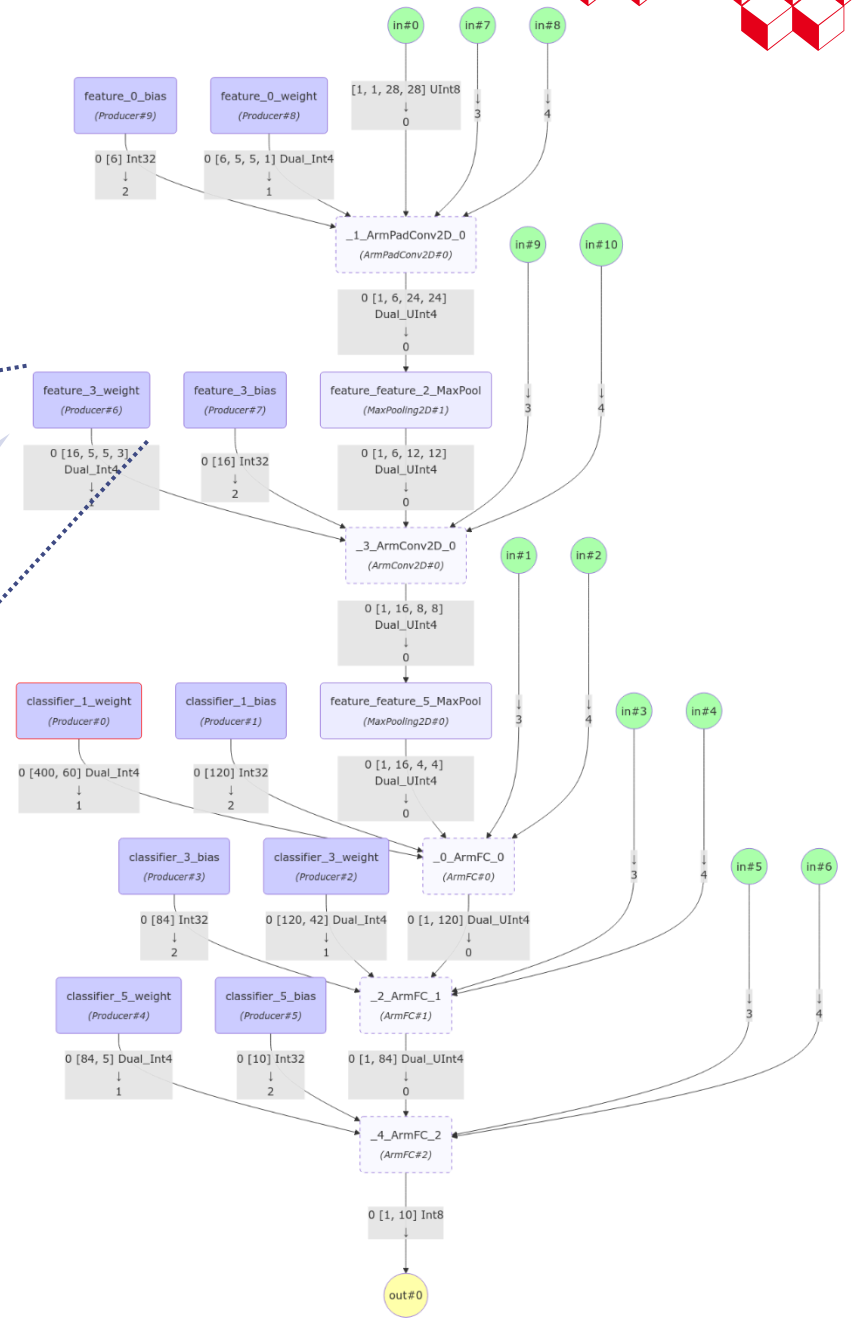
Cast to expected type

Sub-byte memory arrangement



Constant folding is a single operator

`aidge_core.constant_folding(graph)`



Dedicated *ONNX cleaner*

- Graph simplification through pattern matching:
  - Constant/Shape folding
  - MatMul + Add → Gemm
  - Fuse LayerNorm / Gelu w.r.t to Pytorch / Keras variants
  - and many more.... 17 recipes implemented (out of 37 identified)
- **Key features:**
  - 3-tier optimization levels: Training-safe | Accuracy-safe | Performance
  - ONNX Opset compliance verification
  - **Customizable**: select transformations and execution order
  - **Extendable** with your own recipes

```
[15:20:15] > onnx_cleaner --show_recipes
```

Available Graph Transformation Recipes

Recipe	Supported Opsets	Training-Safe	Accuracy-Safe	Brief
ConstantShapeFolding	All	✓	✓	Fold constant and shape node.
FuseGeLU	20-22	✓	✓	Fuse operators to form a GeLU operation.
FuseLayerNorm	17-22	✓	✓	Fuse operator to form a LayerNormalization operation.
FuseMatMulAddToFC	All	✓	✓	Fuse MatMul and Add in a Gemm operator.
FuseBatchNorm	All	✗	✓	Fuse BatchNormalization layer with previous Conv or FC layer.
RemoveIdempotent	All	✓	✓	Remove idempotent nodes that follow each other. Idempotent nodes are: ReLU, Reshape, Ceil, Floor, Round
RemoveIdentity	All	✓	✓	Remove identity nodes from graph, except if it is a graph input with two nodes connected to it.

TensorRT Performance Comparison (1,000 iterations)

Model	Execution Time	Throughput	vs Best
original	0.552 ms	1812.0 inf/s	+0.007 ms (+1.3%)
onnx_cleaner	0.545 ms	1835.4 inf/s	BEST
onnxsim	0.550 ms	1817.1 inf/s	+0.005 ms (+1.0%)
onnxslim	0.554 ms	1806.1 inf/s	+0.009 ms (+1.6%)

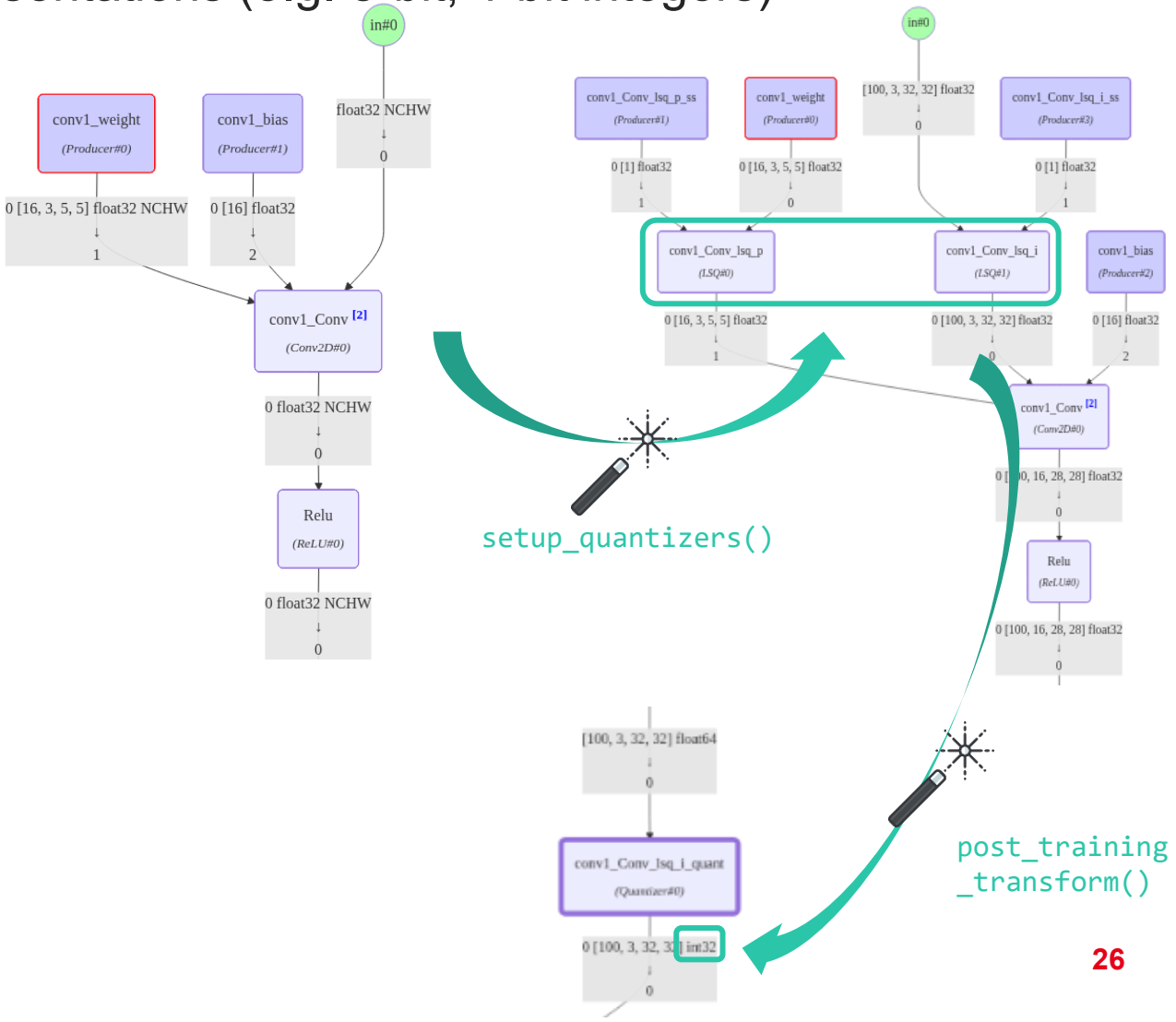
Quantization with PTQ and QAT

- Map high-precision numerical values (e.g. 32-bit floats) of weights, biases, and intermediate activations to lower-precision discrete representations (e.g. 8-bit, 4-bit integers)
  - Quantization-aware training (QAT)
  - Post-training quantization (QAT)

- QAT example:

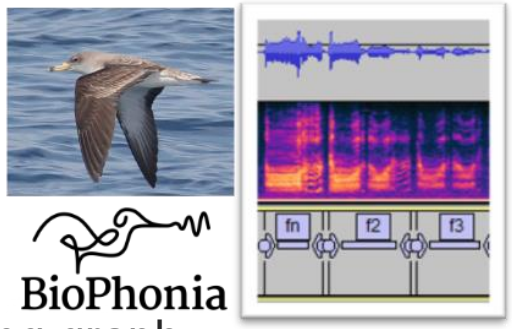
  ***.setup_quantizers()**
  - Automatic quantizers insertion
  - Automatic initialization / parametrization

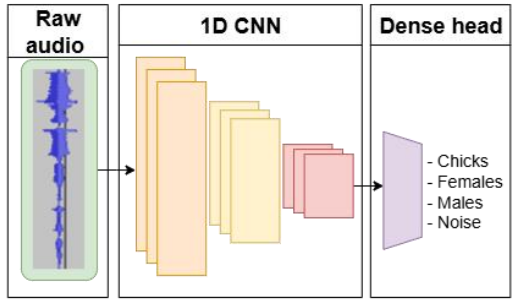
  ***.post_training_transform()**
  - Automatic integer scaling adaptation

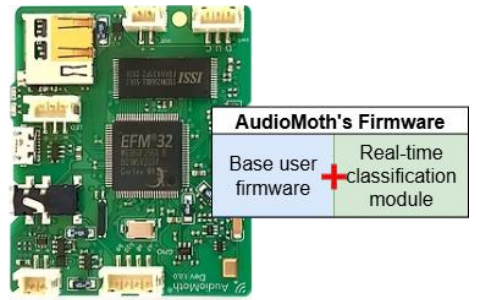


Example of graph tiling: passive acoustic monitoring

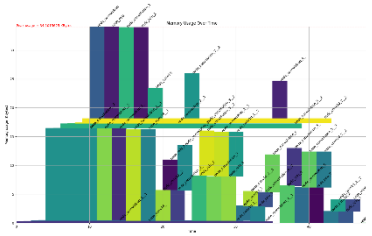
- **Goal**: detect shearwaters call
  - *AudioMoth* hardware target with only 32kB RAM
  - 1D-CNN neural network model

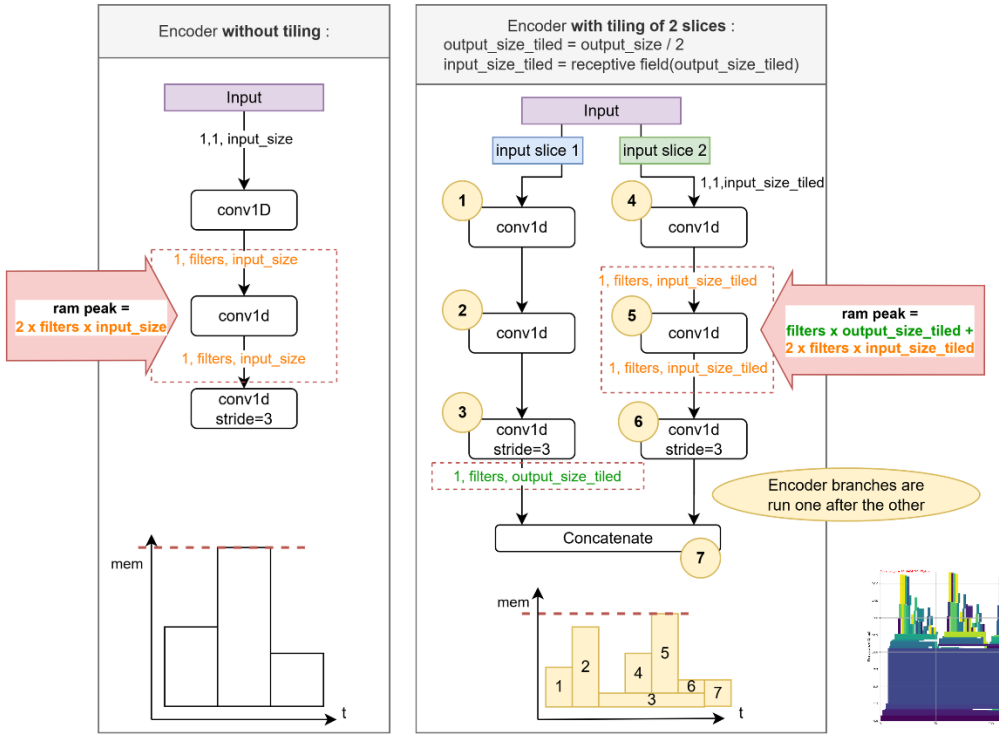


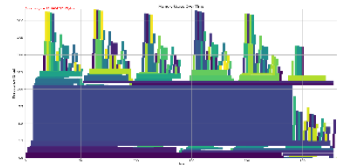


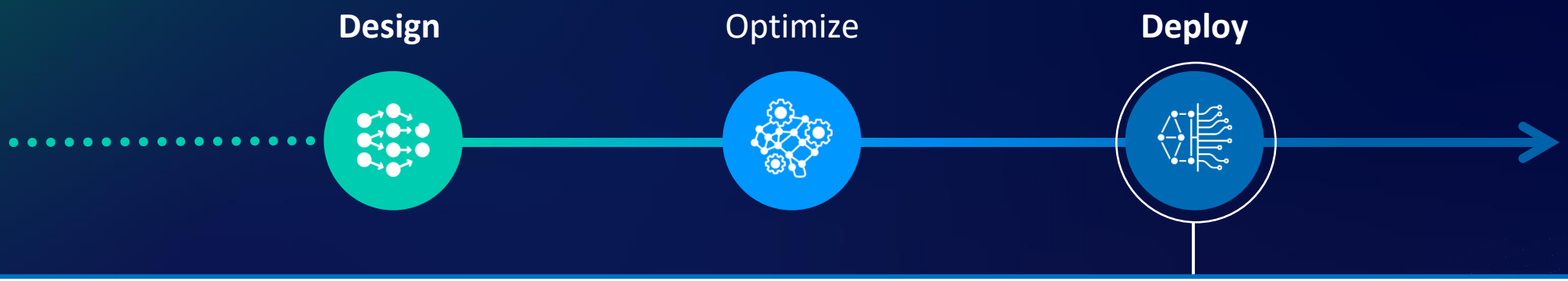


- **Approach**: memory optimization using graph tiling strategy
  - Split encoder output into X slices
  - Compute corresponding input slices
  - Run 1D-CNN encoder on each input slice
  - Concatenate output slices
- **Results**: 25% reduction in memory peak

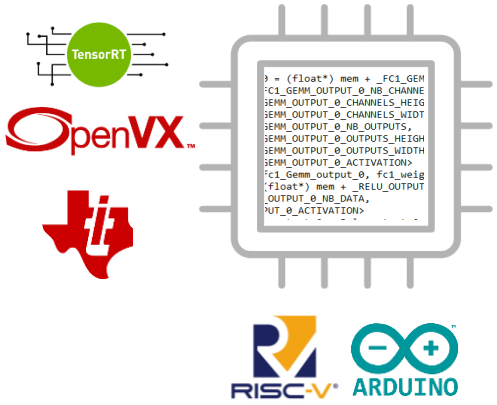








Broad portability



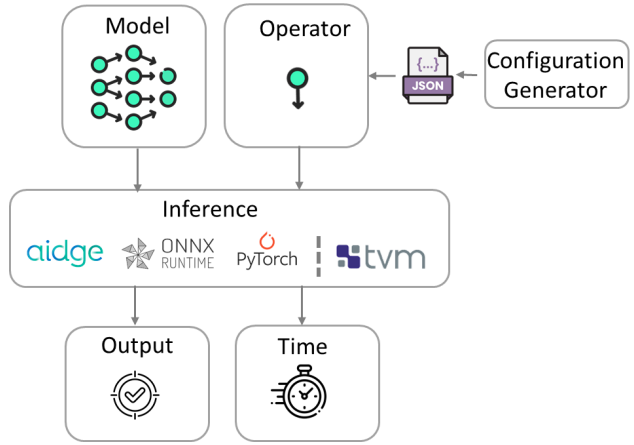


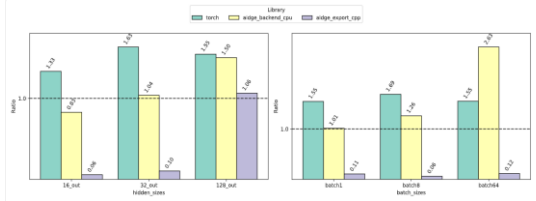
An efficient code generator for multiple hardware targets

Building trustworthy

- **Audit** with open source code
- **Meet strict aeronautical standards**
- Test inference under **hardware fault conditions**
- **Pioneer** in the adoption of the Safety ONNX standard
- **Robust** training

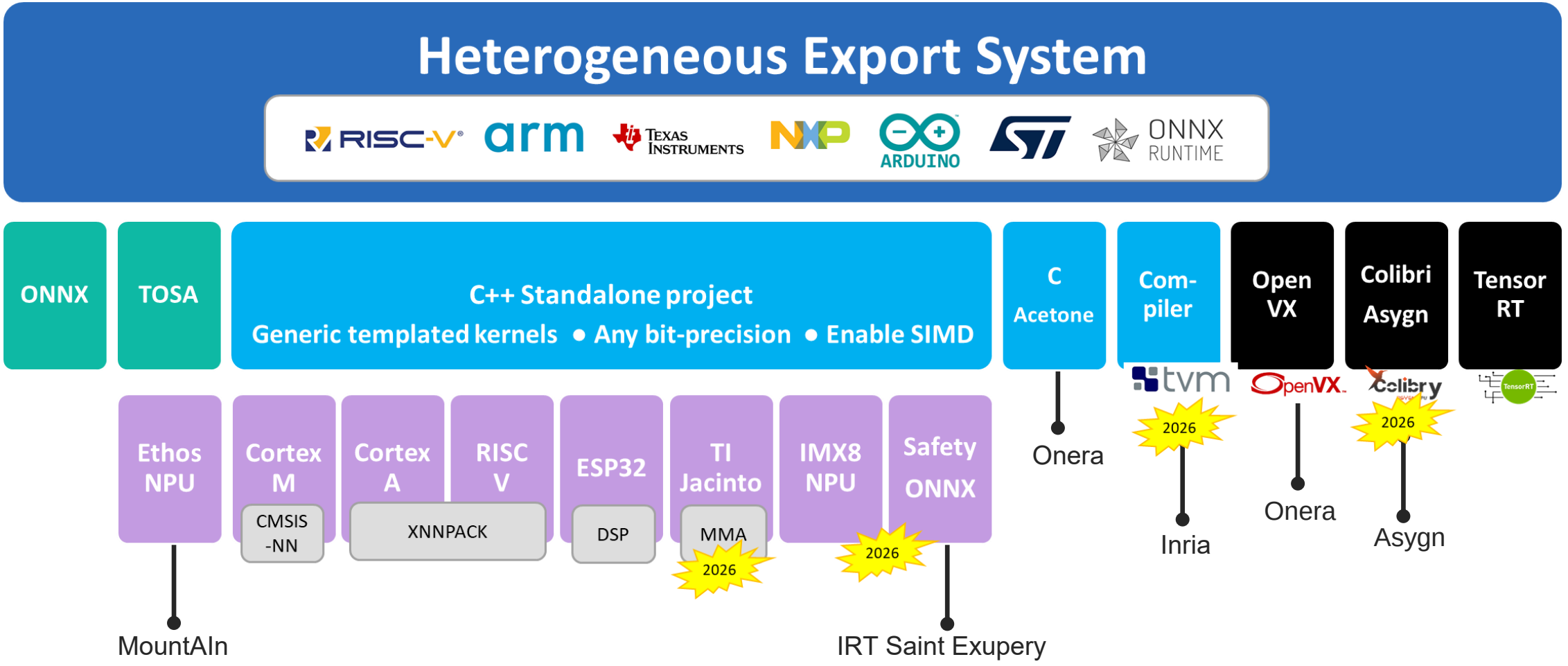
Validate and benchmark





Multi-target export engine

- Overview of code generation engine



Multi-target export engine



Let's go deeper into Aidge's C++ export!

Hardware mapping support

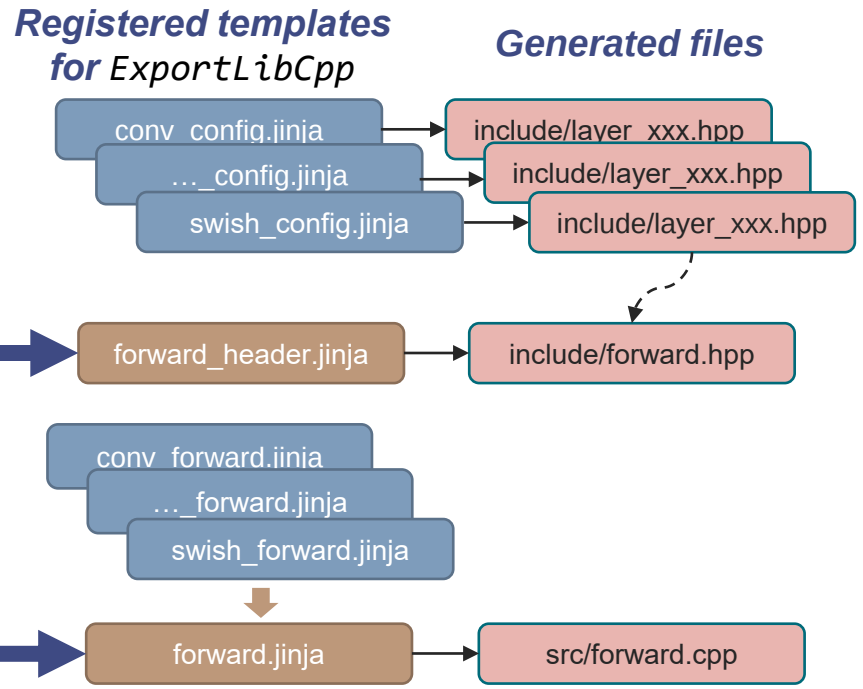
- A flexible code generation engine
  - Template-based (Jinja)
  - Open and transparent (non-binary code generation)
  - Designed to integrate low-level computing primitives

```
model = aidge_onnx.load_onnx("mobilenet.onnx")
model.set_backend("cpu")
model.forward_dims([[1, 3, 224, 224]])

scheduler = aidge_core.SequentialScheduler(model)
scheduler.generate_scheduling()

export_folder = "mobilenet_export"
aidge_core.export_utils.scheduler_export(
        scheduler,
        export_folder,
        aidge_export_cpp.ExportLibCpp,
        memory_manager=aidge_core.mem_info.generate_optimized_memory_info,
        memory_manager_args={"stats_folder": f"{export_folder}/stats"}
)

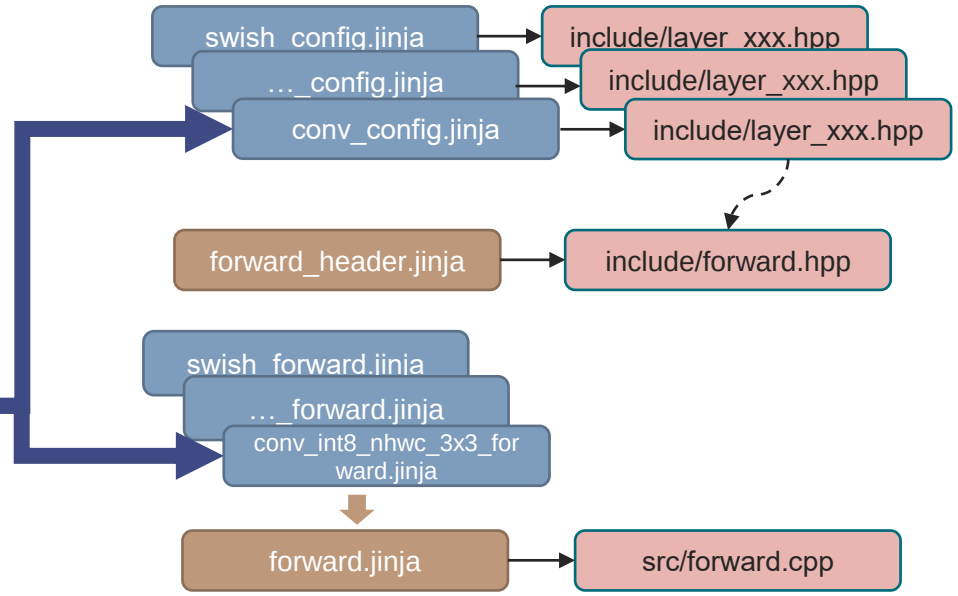
aidge_core.export_utils.generate_main_cpp(export_folder, model)
```



Hardware mapping support

- Kernels are registered **with a specification:**
  - Required I/O data type, data format, dimension range, attributes

```
@aidge_export_cpp.ExportLibCpp.register("Conv",
    aidge_core.ImplSpec(
        [ # Inputs specs
            aidge_core.IOSpec(aidge_core.dtype.int8, aidge_core.dformat.nhwc),
            aidge_core.IOSpec(aidge_core.dtype.int8, aidge_core.dformat.nhwc,
[[0,0], [0,0], [3,3], [3,3]]),
            aidge_core.IOSpec(aidge_core.dtype.int32)],
        [ # Outputs specs
            aidge_core.IOSpec(aidge_core.dtype.int32)],
        { # Attributes specs
        }))
class ConvCpp(aidge_core.export_utils.ExportNodeCpp):
    def __init__(self, node, mem_info):
        super().__init__(node, mem_info)
        self.config_template = "conv_config.jinja"
        self.forward_template = "conv_int8_nhwc_3x3_forward.jinja"
        self.include_list = []
        self.kernels_to_copy = [
            "conv_int8_nhwc_3x3_kernel.hpp",
        ]
```



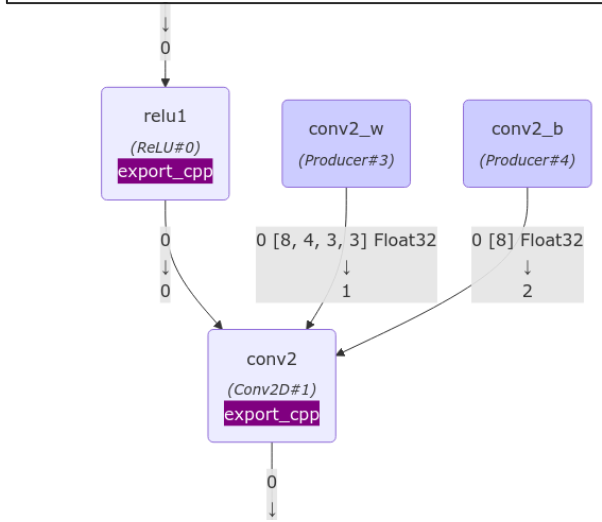
Register multiple specialized kernels for the same operator with different specifications!

Here: INT8 NHWC convolution kernel of size 3x3 only

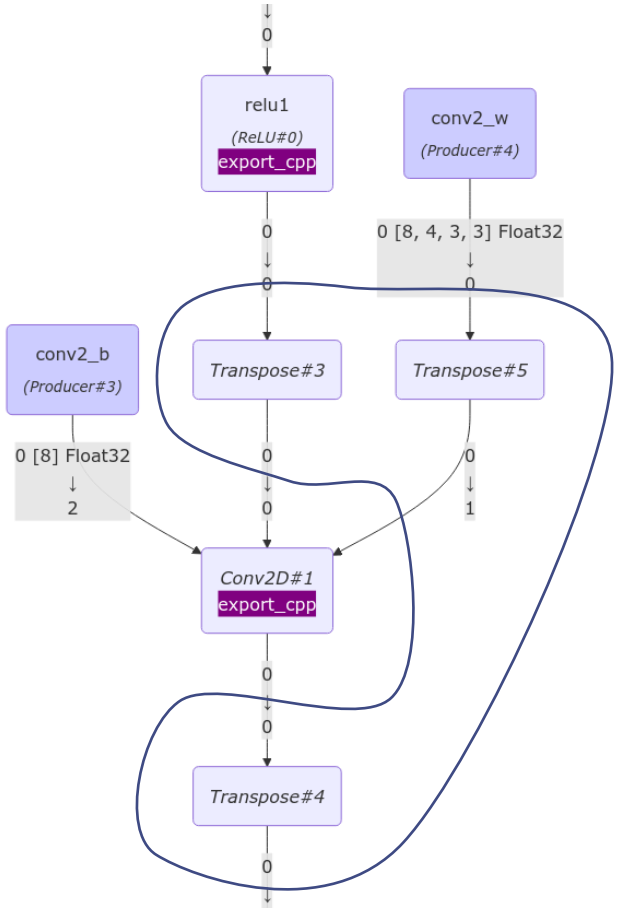
Hardware mapping support

- Specifications allow **automatic graph adaptation**
  - To use available computational kernels

```
graph.set_backend("export_cpp")
aidge_core.adapt_to_backend(graph)
```



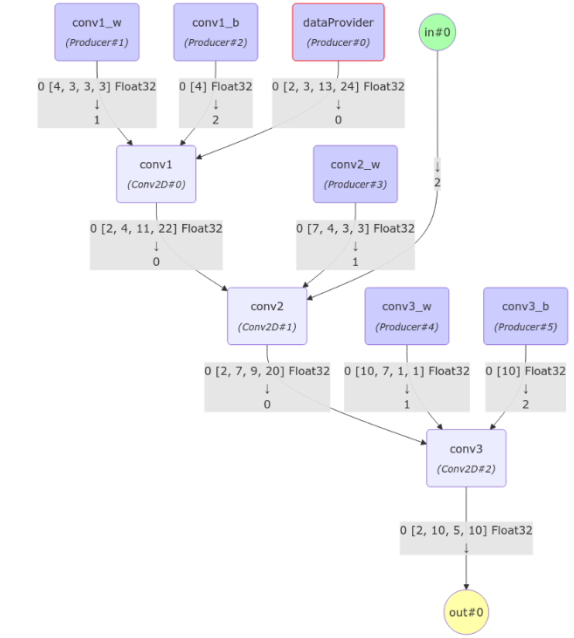
Simple example with data format change (NCHW ➔ NHWC)



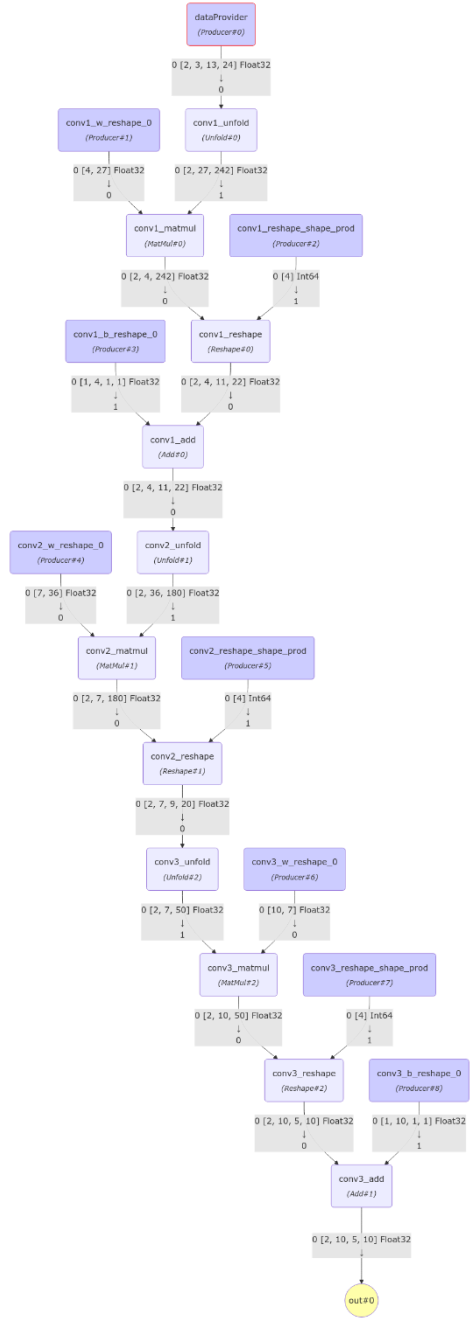
Hardware mapping support

- Specifications allow **automatic graph adaptation**
  - To adapt the implementation with respect to the algorithm

```
aidge_core.conv_to_matmul(graph)
```



Replace Conv with Unfold + MatMul operators



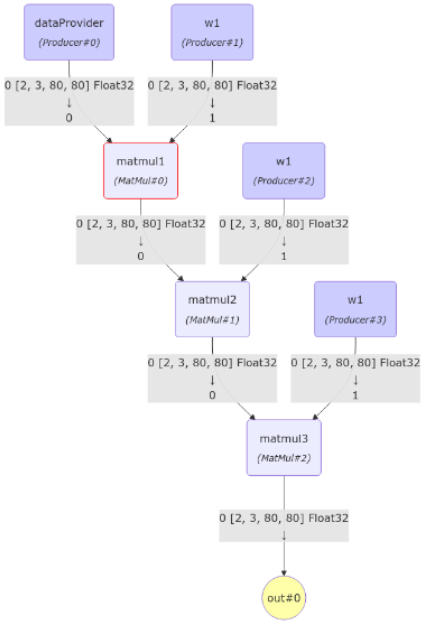
Hardware mapping support

- Specifications allow **automatic graph adaptation**
  - To match the hardware architecture
    - Tiling MatMul

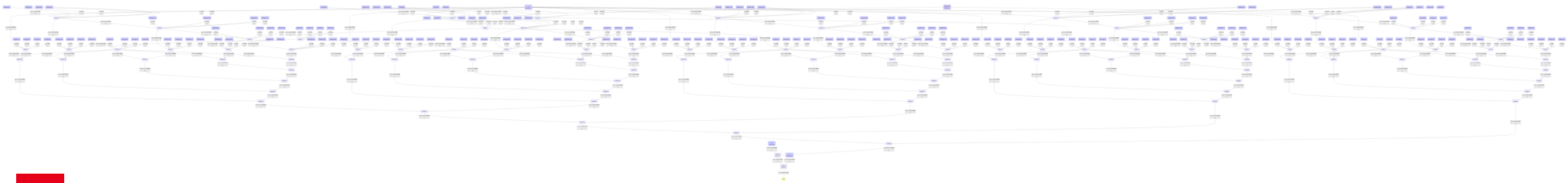
```
aidge_core.matmul_tiling(matMul, [16, 16])
```

    - Tiling Conv2D

```
aidge_core.get_conv_horizontal_tiling(conv, axis=1,
nb_slices=5)
```



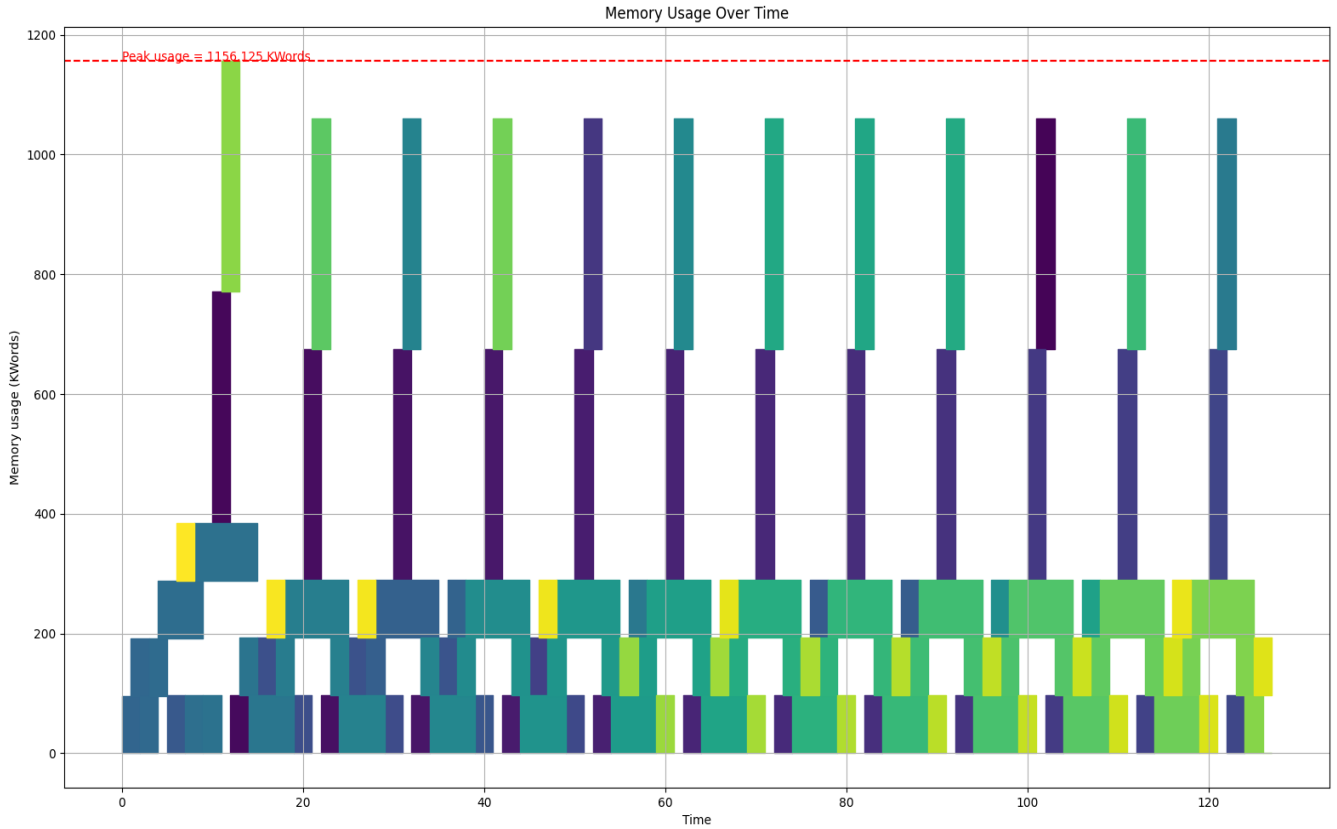
Example: Fixed size MatMul



Hardware mapping support

- **Static memory allocation** for generated code with a memory manager
  - Memory reuse: in-place or partially in-place (wrapping)
  - Auto-concatenation (strided buffers support)
  - Several optimization strategies available

```
scheduler = aidge_core.SequentialScheduler(dinov2_model)
scheduler.generate_scheduling()
aidge_core.generate_optimized_memory_info(scheduler,
"stats_dino", wrapping=False)
```



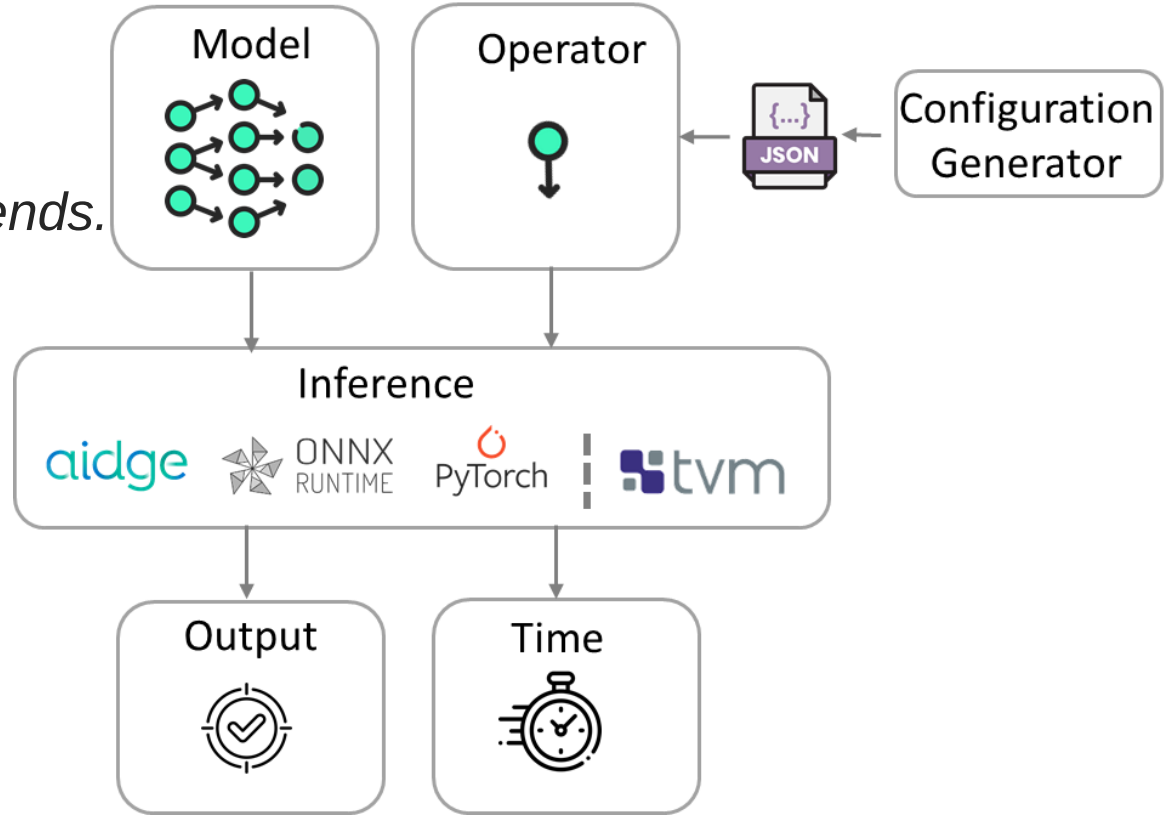
Example of export: Visual-Based Runway Detection

- **Goal**: perform the aeronautics use-case of runway detection;
- **Approach**: code generation from a pre-trained detection model using Aidge's export module ACETONE.
  - Use a pre-trained YOLOv8 model (from the LARD project);
  - Run inference with PyTorch and ORT for benchmark;
  - Export and import ONNX file within Aidge;
  - Generate standalone C code;
- **Results:** a certifiable generated C code, which:
  - Preserves the inference model semantics;
  - Guarantees the requirements traceability → mapping between model description, C code and binary file (-O0 flag);
  - Is timing predictable → WCET bounds can be derived with OTAWA;
  - Enforces efficient resources usage.



Performance benchmark module

- **Goals**
  - Measure and compare average inference time;
  - Validate numerical output precision across *backends.*
- **Evaluation scope**
  - Full end-to-end model benchmarking;
  - Operator-level profiling with customizable configurations.
- **Backend and runtime coverage**
  - Aidge backend CPU or CUDA or any *export;*
  - External runtimes (ONNX-RT, PyTorch, TVM, ST Edge AI) .
- **Hardware integration**
  - Dedicated hardware-in-the-loop (HIL) testing Setup.




```
conda:(aidge) cm264821@is156025 /data1/is156025/cm264821/aidge_packages/dev/1-experiment/onnx
[14:22:12] > compare_ort_aidge conv1d_model_torch.onnx
ONNX Runtime vs Aidge Inference Comparison for conv1d_model_torch
```

Edge Name	ONNX Runtime	Aidge	Equal	Avg Error
conv1d_Conv_output_0	✅	✅	✅	4.8894e-09
output	✅	✅	✅	9.3132e-10

```
Comparison Summary for conv1d_model_torch (ONNX Runtime vs Aidge)
ONNX Runtime edges with no Aidge is_equal: 0 / 2 (0.00e+00%)
Aidge edges with no ONNX is_equal: 0 / 2 (0.00e+00%)
Matching outputs: 2
Mismatching outputs: 0
```

Use cases enabled by aidge

Defect Detection and Classification

ArcelorMittal

- Vision-based model for small defect detection (~mm)
- Enabling high speed processings (20m/s)
- Deployed on Nvidia GPU



Passive Acoustic Monitoring

BioPhonia

- Extreme lightweight model for shearwaters sound classification
- **Reduce by 50% the peak memory**
- Deployed on Audiomoth / Silicon Labs



Heat Pump Monitoring

- **Lightweight prediction algorithm** for adaptive heat pump control
- Up to **40% energy saving** for collective housing
- Deployed on STM32

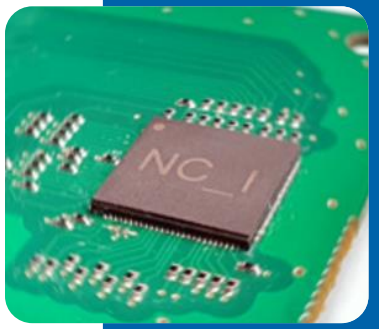
EDF UGA Université Grenoble Alpes cea



Hardware Design : NeuroCorgi AI accelerator

- RTL generation of quantized model
- HD images processing in real time : **latency is less than 10ms**
- Uses **1,000 times less power** than commercial circuits

cea



cea

Use cases enabled by aidge

Visual Based Landing

- Runway detection
- Code generation to **meet certification requirements** (traceable and predictable)
- Deployment on Texas Instruments

ONERA THE FRENCH AEROSPACE LAB AIRBUS CS GROUP



Decision-making support

- **Sovereign** (no regulation restriction) **and whitebox tool**
- Ensure **bit-accurate results** with simulation
- **Competitive performance** compared to proprietary tools

MBDA



Welding defect detection in pipes

- **Classify images as pristine or defective**
- More than 75% of pristine images removed from human analysis.
- Quantized and exported on a microcontroller iMX8

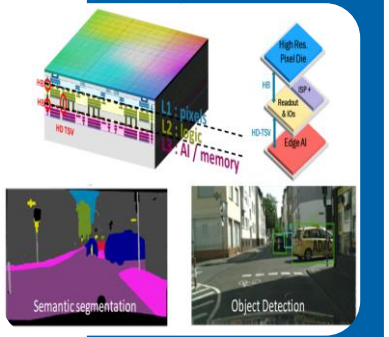
cea



Smart sensing applications

- Using **3D stack image sensor**
- Model quantization and NPU code generation
- Up to **3× fewer MAC units and 3× less silicon area** compared to state-of-the-art designs

NANOELEC. STMicroelectronics cea



Conclusion and perspectives

- **What we saw**
  - A unified toolchain to load a model (e.g. ONNX format), optimize it and generate associated deployable code;
  - A transparent and searchable IR that is scalable to answer the constraints of hardware implementations;
  - An easily extendable framework to implement state-of-the-art optimization methods and leverage third-party libraries capacities.
- **Upcomming features**
  - Code generation for heterogeneous (library + hardware) targets;
  - Quantization for Transformer base model;
  - Support for more NN models and hardware targets.



Join us

aidge

Gitlab

Code repository, issues, and discussions

https://gitlab.eclipse.org/eclipse/aidge

News

Follow us on LinkedIn

for the latest news and updates

Documentation

User guide, API and +15 tutorials

https://eclipse.dev/aidge/

Hugging Face

Some reference models

https://huggingface.co/EclipseAidge

Wiki

Contributor onboarding

https://gitlab.eclipse.org/groups/eclipse/aidge/-/wikis/

Chat

Engage with the community

https://chat.eclipse.org/#/room/#aidge:matrix.eclipse.org

aidge

Hands-on session

1) Go to *https://gitlab.eclipse.org/eclipse/aidge*
2) Navigate to `aidge → examples → tutorials → 101_first_step`

