aidge

An Eclipse Foundation Project

Enabling Open-Source Edge AI for Software-Defined Vehicles

DeepGreen NeuroKit2E

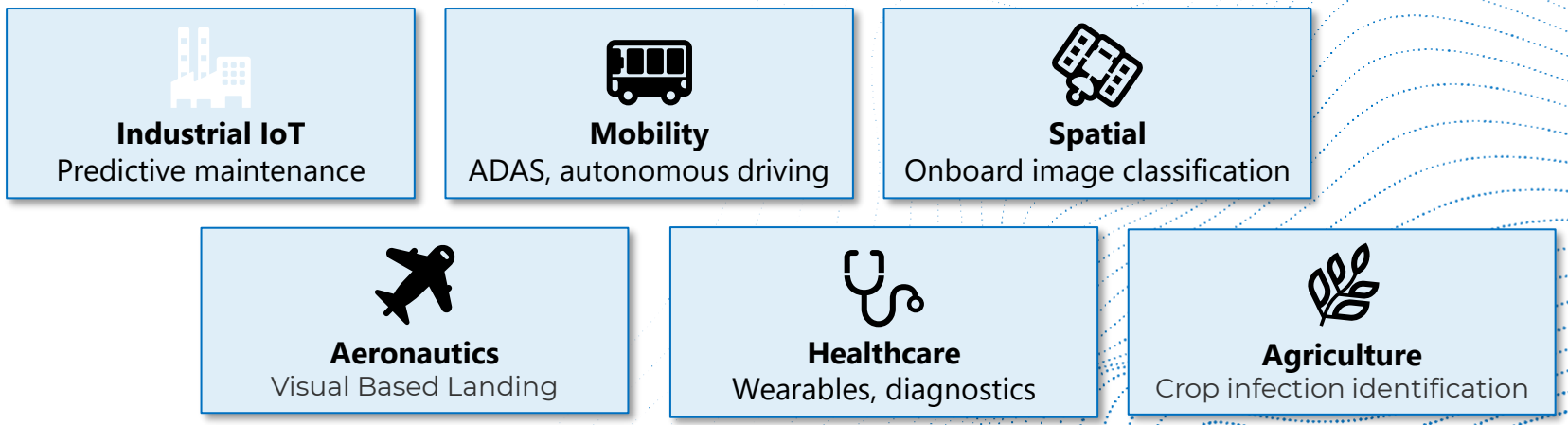
Edge AI revolution

Integrate AI capabilities directly where data is produced

⚡ Key benefits

- **Real-time processing**: ultra-low latency inference
- **Privacy protection**: data and model stay on device
- **Offline operation**: no internet dependency
- **Cost reduction**: energy consumption and infrastructure

🎯 Many aplications



Edge AI revolution

SDV and key use cases for embedded AI

Autonomous Driving & ADAS
Real-time perception, steering, braking, lane keeping

Predictive Maintenance
Early fault detection, reduced downtime

Personalized In-Car Experience
Adaptive comfort, infotainment, driver preferences

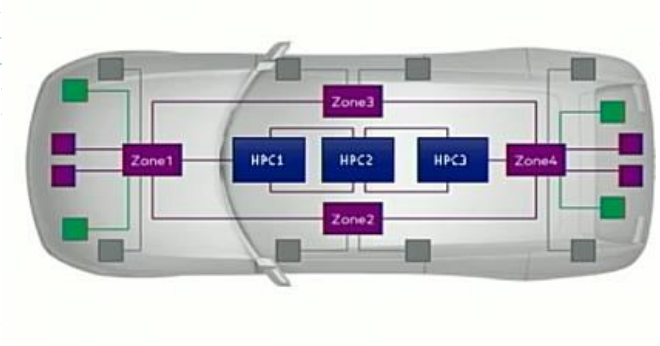
Voice & Gesture Control
Natural interaction with vehicle systems

Driver Monitoring & Safety
Fatigue detection, emergency response

Edge AI revolution

Evolution of embedded computing for mobility

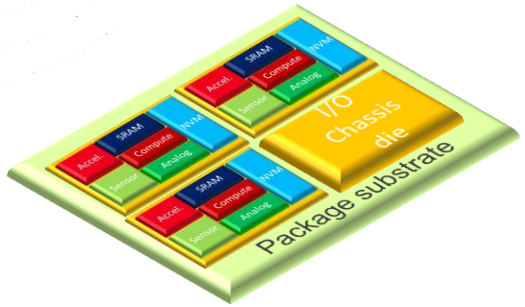
Centralized & zonal electronic architecture



Expected benefits

- Reduce weight and volume of ECUs
- ECUs resources sharing
- Upgrade by SW and I/0 rather than additional ECU

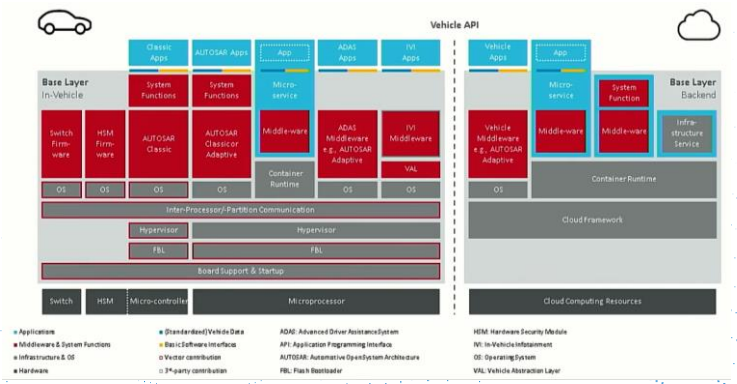
Chiplet-based SoC



Expected benefits

- Enable heterogeneous computing
- Integration of domain-specific accelerators
- Sustainable and re-usable

Architecture SW & DevOps



Vector, SIA CESA, Dec. 2022

Expected benefits

- Decouple HW and SW SOP
- Enable update thanks to full FOTA
- Enable the upgrade on demand

Global co-design is required

Application ↔ Algorithm ↔ Software ↔ Data coding ↔ Architecture ↔ Technology

Structural limitations of edge AI 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

Needs for edge AI

Streamlining the workflow

- Ensure smooth interoperability across tools and toolchains
- Manage hardware heterogeneity end-to-end (from component to system)

Mastering the deployment

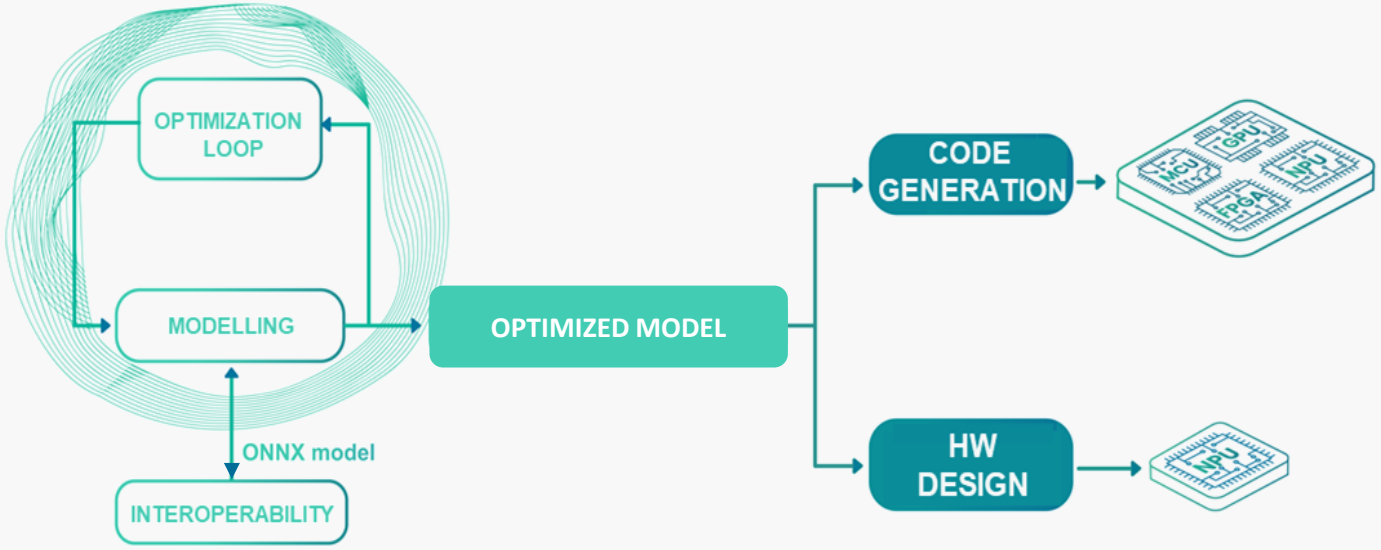
- Provide controlled, scalable, and open optimization and deployment pipeline
- Integrate sovereign technologies (chiplet architectures, advanced memory) for sustained innovation

Certification readiness

- Meet the stringent expectations of safety-critical environments
- Accelerate compliance with industry standards and regulatory frameworks

aidge

Edge AI deployment framework



ONNX

PyTorch TensorFlow K

API

+

Multi-Platform and Packaging

Windows Mac ubuntu PIP

Beyond fragmentation & black-box

End-to-end integrated platform

- From design to optimized deployment
- High degree of interoperability

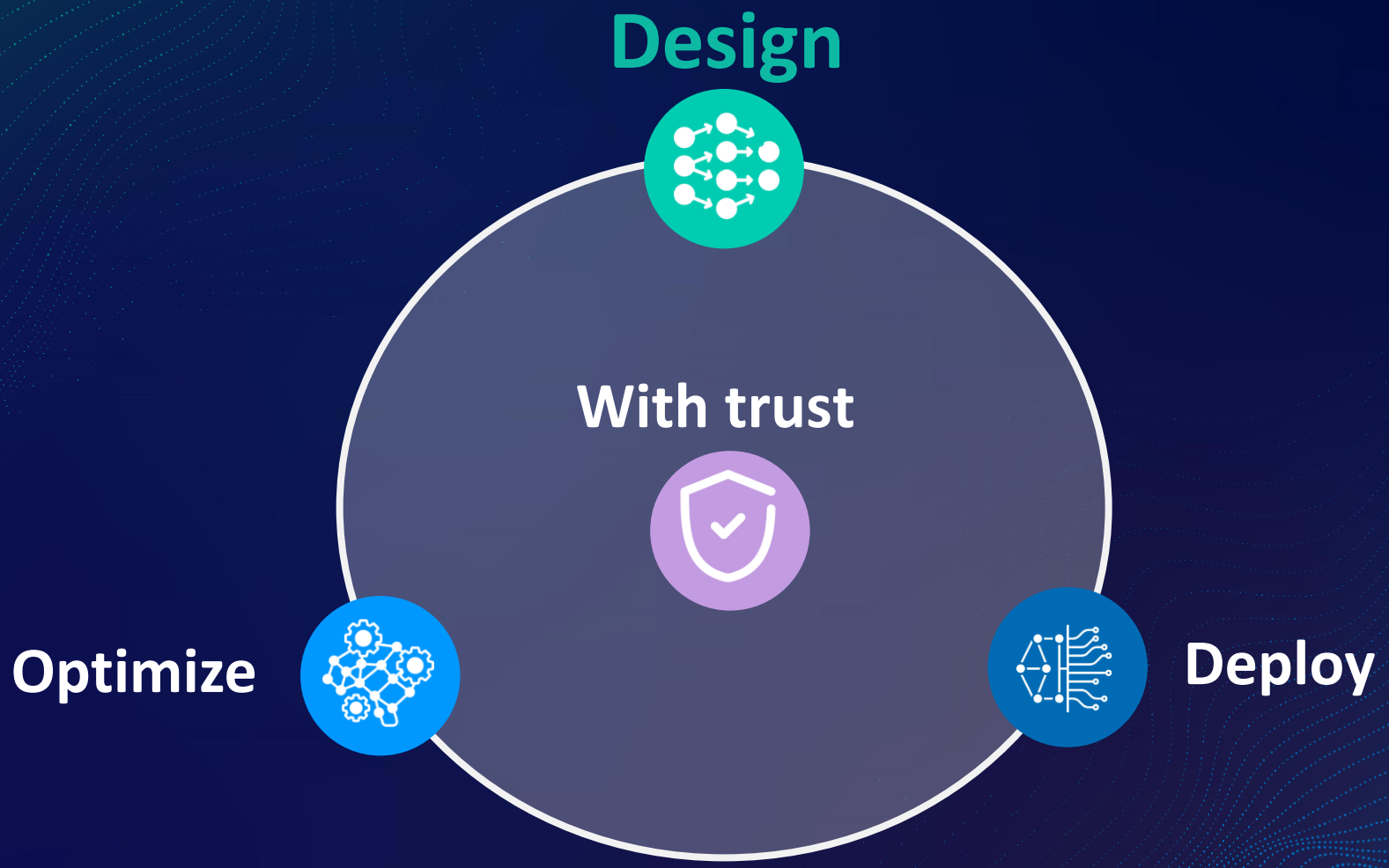
Open and modular platform

- Lightweight core module with plug-ins
- Minimal dependencies
- Collaborative environment

Hosted by

ECLIPSE FOUNDATION

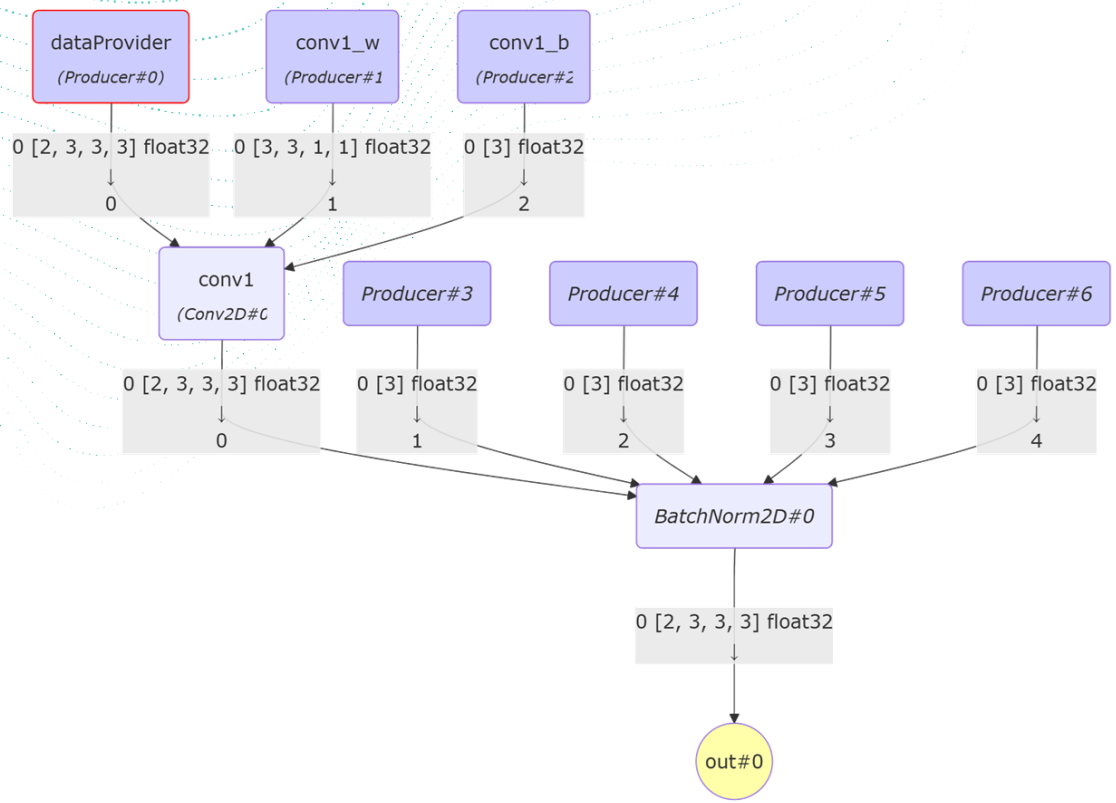
Complete framework with innovations

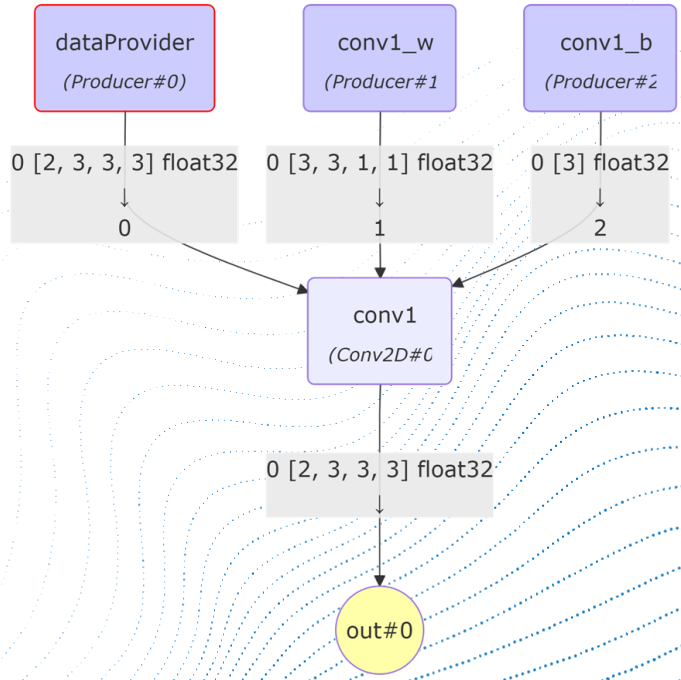


aidge

Great design begins with a great intermediate representation

A simple example: match and fuse two operators





The right IR: match and fuse two operators

Can we do it directly with only ONNX?

```python
class ConvBatchNormSolution(Solution):
    def __init__(self, begin, begin_n, end_p, end):
        Solution.__init__(self, begin, begin_n, end_p, end)

    def apply(self, node_list):
        if self.end_p.is_reserved:
            return None, False

        [conv_weight_name, conv_weight_initilizer, conv_bias_name, conv_bias_initilizer]
              = fuse(begin, begin_n, end_p, end)

        self.begin_n.in_redirect(self.begin_n.origin.input[1], conv_weight_name)
        if len(self.begin_n.input) > 2:
            self.begin_n.in_redirect(self.begin_n.origin.input[2], conv_bias_name)
        else:
            self.begin_n.input[conv_bias_name] = conv_bias_name
        self.begin_n.initializers = [conv_weight_initilizer, conv_bias_initilizer]

        self.begin_n.successor = []
        for end_ in self.end:
            end_.in_redirect(self.end_p.origin.output[0], self.begin_n.origin.output[0])
            self.begin_n.successor.append(end_)
            end_.precedence[end_.precedence.index(self.end_p)] = self.begin_n

        node_list.remove(self.end_p)

        return node_list, True
```

Required sub-class

```python
def find_and_fuse(node):
        if (
            node.origin.op_type == "Conv"
            and len(node.successor) == 1
            and node.successor[0] is not None
        ):
            if len(node.initializers) > 0:
                return None
            next = node.successor[0]
            if next.origin is not None and next.origin.op_type == "BatchNormalization":
                if len(node.initializers) > 0:
                    return None
                if len(node.get_precedence_by_idx(1).tensors) == 0:
                    return None
                elif (
                    len(node.precedence) > 2
                    and len(node.get_precedence_by_idx(1).tensors) == 0
                ):
                    return None
                else:
                    for idx_ in range(1, 5):
                        if len(next.get_precedence_by_idx(idx_).tensors) == 0:
                            return None

                solution = ConvBatchNormSolution(
                    node.get_precedence_by_idx(0), node, next, next.successor
                )
                return solution
```

A nightmare...
Extremely low level code!

Source:
https://github.com/microsoft/onnxconverter-common/blob/master/onnxconverter_common/optimizer.py

The right IR: match and fuse two operators

Maybe ONNX Script can help?

```python
def _reshape_for_broadcast(x: np.ndarray, rank: int, axis: int = 1) -> np.ndarray:
    # Build shape: 1s everywhere except -1 at the target axis
    broadcast_shape = [1 if axis != i else -1 for i in range(rank)]
    return np.reshape(x, broadcast_shape)


class _FuseBatchNormBase(RewriteRuleClassBase, ABC):
    @abstractmethod
    def get_filters_axis(self, attributes: Mapping[str, ir.Attr]) -> int:

    def rewrite(self, op, x: ir.Value, inbound_out: ir.Value, batchnorm_out: ir.Value):
        return fuse(inbound_out, batchnorm_out)

    def check(self, context, x, inbound_out: ir.Value, batchnorm_out: ir.Value) ->
MatchResult:
        del context  # Unused
        check_result = MatchResult()

        inbound_node = inbound_out.producer()
        batchnorm_node = batchnorm_out.producer()

        # Check that inbound weights + (inbound bias) + batchnorm params are initializers
        # and that they are not graph inputs
        initializers = [inbound_node.inputs[1], *batchnorm_node.inputs[1:]]
        if len(inbound_node.inputs) > 2:
            initializers.append(inbound_node.inputs[2])

        for initializer in initializers:
            if not initializer.is_initializer() or initializer.const_value is None:
                return check_result.fail(f"{initializer.name} is not a constant
initializer.")
            if initializer.is_graph_input():
                return check_result.fail(f"{initializer.name} is a graph input.")

        return check_result
```

```python
class FuseBatchNormIntoConv(_FuseBatchNormBase):
    op_type: ClassVar = "Conv"

    def get_filters_axis(self, attributes: Mapping[str, ir.Attr]) -> int:
        return 0

    def pattern(self, op, x):
        return op.BatchNormalization(
            op.Conv(x, _allow_other_inputs=True, _outputs=["inbound_out"]),
            _allow_other_inputs=True,
            _outputs=["batchnorm_out"],
        )
```

Pattern is a graph: quantifiers or placeholders are not possible
Need for specific attributes to handle edges capture

Required sub-class

Need to specify extra topological checks for the matching

Source:
https://github.com/microsoft/onnxscript/blob/main/onnxscript/rewriter/rules/common/_fuse_batchnorm.py

The right IR: match and fuse two operators

How about *experimental* PyTorch FX?

```python
def matches_module_pattern(pattern: Iterable[Type], node: fx.Node,
modules: Dict[str, Any]):
    if len(node.args) == 0:
        return False
    nodes: Tuple[Any, fx.Node] = (node.args[0], node)
    for expected_type, current_node in zip(pattern, nodes):
        if not isinstance(current_node, fx.Node):
            return False
        if current_node.op != 'call_module':
            return False
        if not isinstance(current_node.target, str):
            return False
        if current_node.target not in modules:
            return False
        if type(modules[current_node.target]) is not expected_type:
            return False
    return True
```

```python
def _parent_name(target : str) -> Tuple[str, str]:
    """
    Splits a ``qualname`` into parent path and last atom.
    For example, `foo.bar.baz` -> (`foo.bar`, `baz`)
    """
    *parent, name = target.rsplit('.', 1)
    return parent[0] if parent else '', name

def replace_node_module(node: fx.Node, modules: Dict[str, Any],
new_module: torch.nn.Module):
    assert(isinstance(node.target, str))
    parent_name, name = _parent_name(node.target)
    setattr(modules[parent_name], name, new_module)
```

```python
def find_and_fuse(model: torch.nn.Module) -> torch.nn.Module:
    model = copy.deepcopy(model)
    fx_model = fx.symbolic_trace(model)
    modules = dict(fx_model.named_modules())
    new_graph = copy.deepcopy(fx_model.graph)
    for node in new_graph.nodes:
        if matches_module_pattern([(nn.Conv2d, nn.BatchNorm2d)], node,
modules):
            if len(node.args[0].users) > 1
                # Output of conv is used by other nodes
                continue
            conv = modules[node.args[0].target]
            bn = modules[node.target]
            fused_conv = fuse(conv, bn)
            replace_node_module(node.args[0], modules, fused_conv)
            node.replace_all_uses_with(node.args[0])
            new_graph.erase_node(node)
    return fx.GraphModule(fx_model, new_graph)
```

Need to contruct an internal formalized graph representation first

Need to write ones own pattern matcher!

Extra check required

Required boilerplate

Sources:
https://github.com/pytorch/pytorch/blob/main/torch/fx/experimental/optimization.py
https://pytorch-cn.com/tutorials/intermediate/fx_conv_bn_fuser.html

The right IR: match and fuse two operators

Make it simple with aidge

An innovative regular-expression like graph matching DSL

```
def find_and_fuse(model: aidge_core.GraphView):
    gm = aidge_core.SinglePassGraphMatching(model)
    matches = gm.match("Conv2D->BatchNorm2D#;(BatchNorm2D#<*-Producer)*");

    for match in matches:
        conv_node = match.graph.root_node();
        bn_node = match.anchors["BatchNorm2D"]["#"]
        fused_conv = fuse(conv_node, bn_node);
        aidge_core.GraphView.replace(match.graph, fused_conv);
```

A transparent, well-defined and explicit graph representation

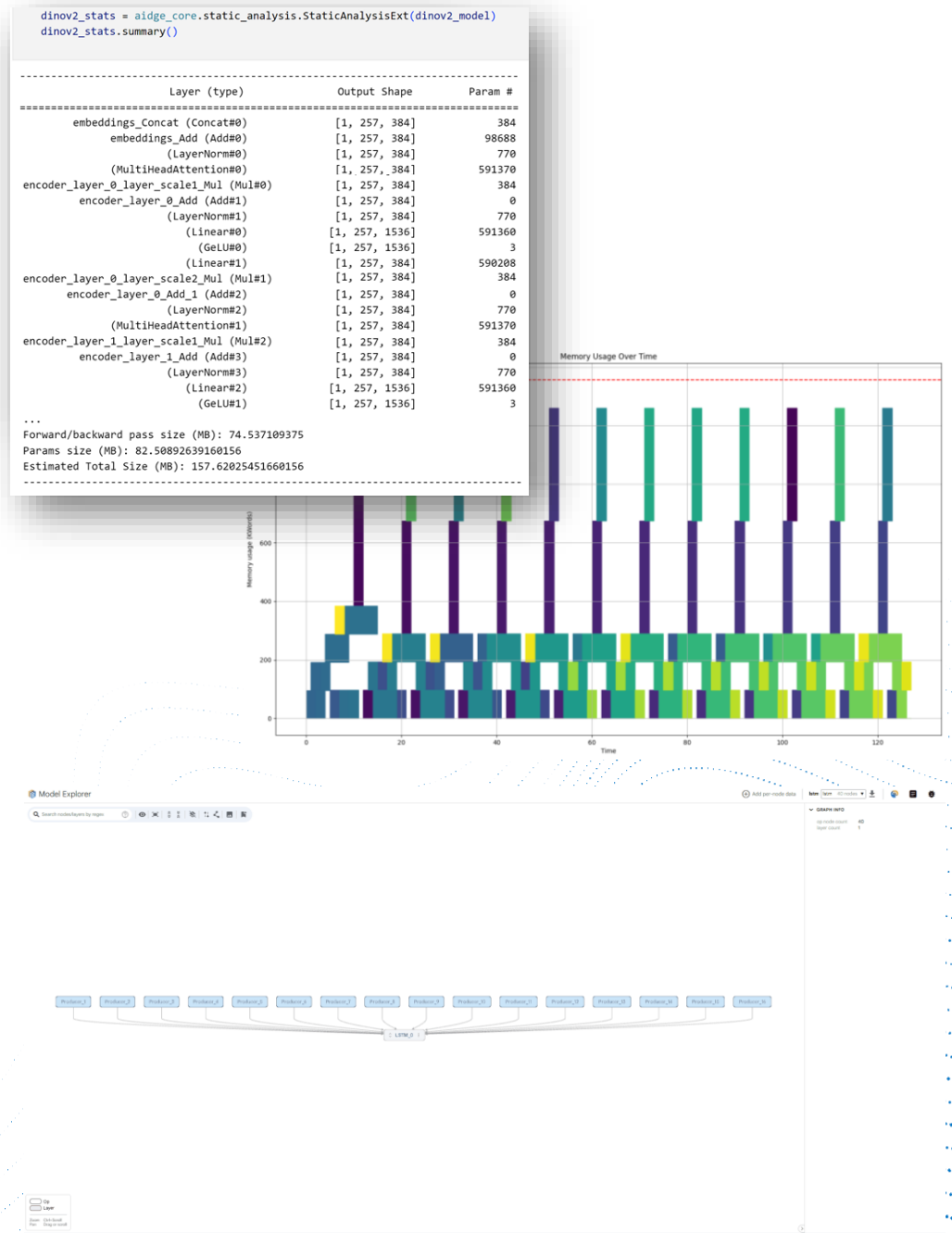
aidge

Deep insights for better decision

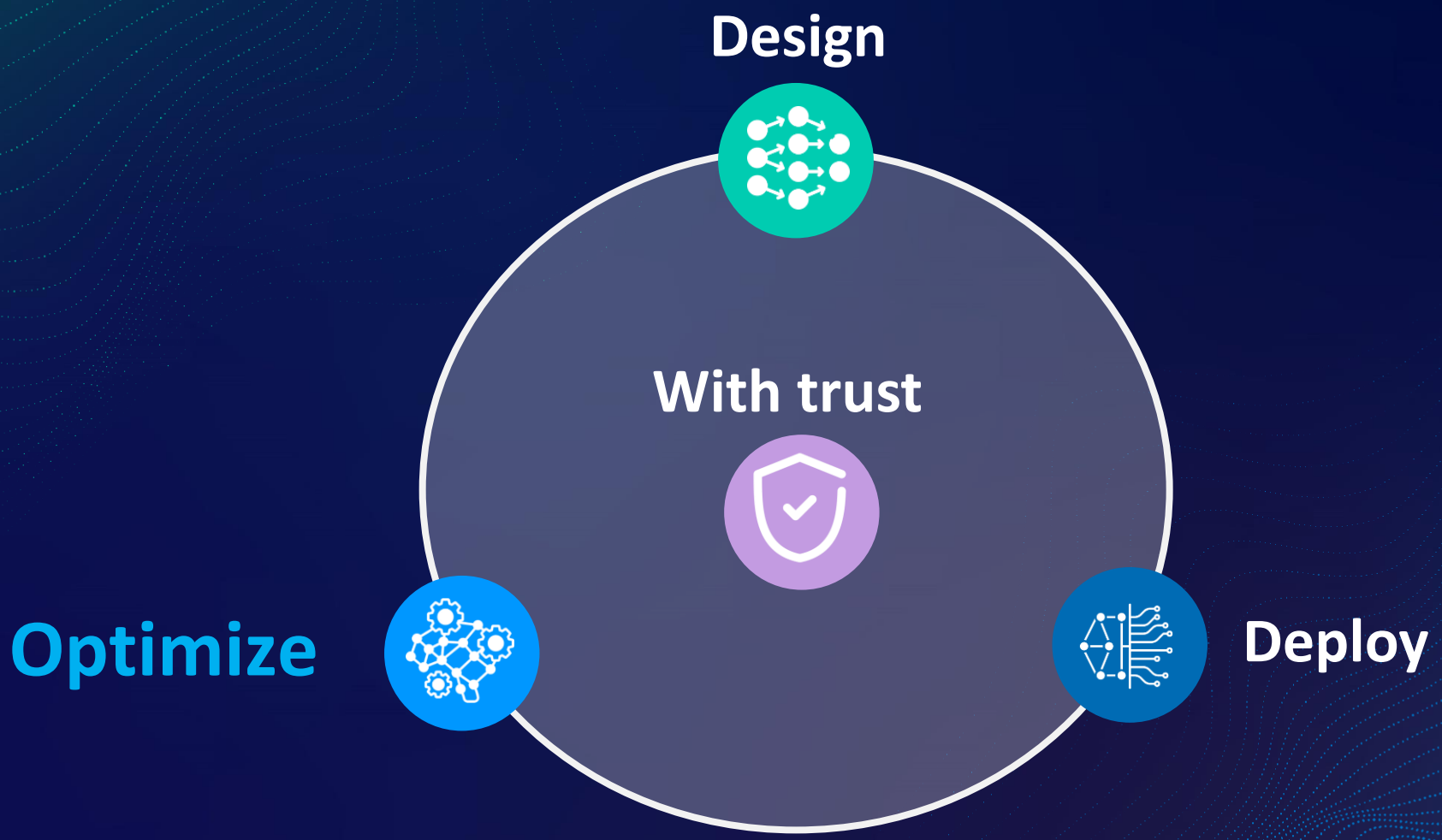
- **Understand and optimize your models**: leverage rich analysis tools for deep understanding and improvement of your models' complexity and resource footprint.

- **Dive into model analysis**: "Aidge Model Explorer" empowers you to intuitively navigate your model architectures (based on Google Model Explorer).

- **Benchmark with confidence**: easily compare the performance of Aidge (C++, CUDA) with other frameworks (PyTorch, ONNX).

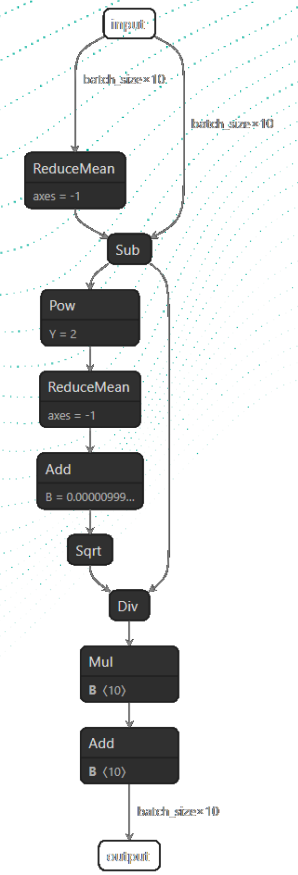


Complete framework with innovations

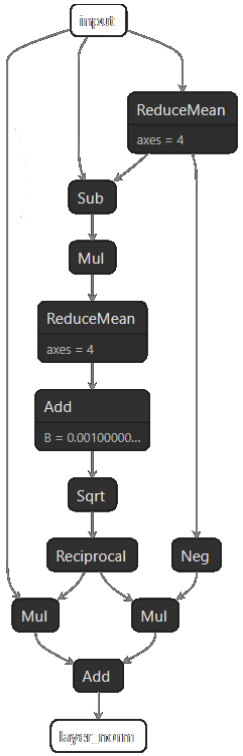


Automatically simplify ONNX graphs

Example with LayerNorm fusion



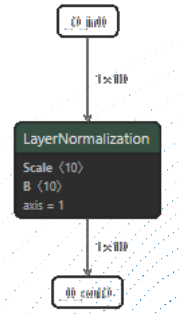
or



Exported from **PyTorch**

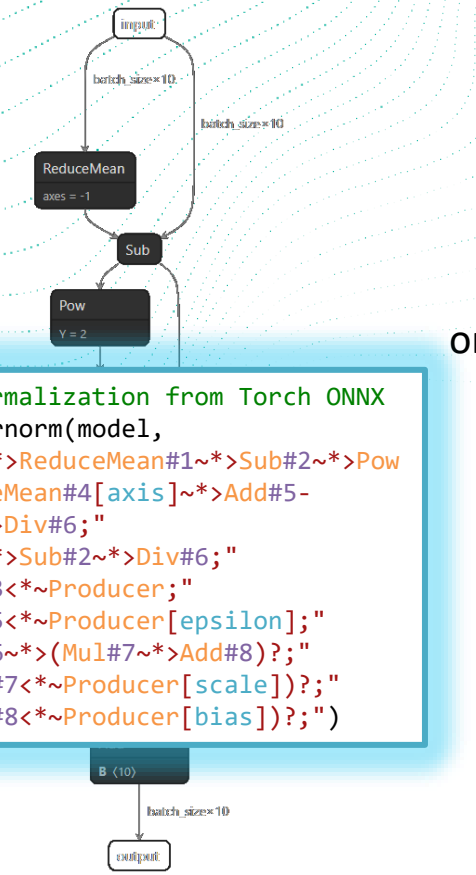
Exported from **Keras**

How to handle complex, non canonical sub-graphs?



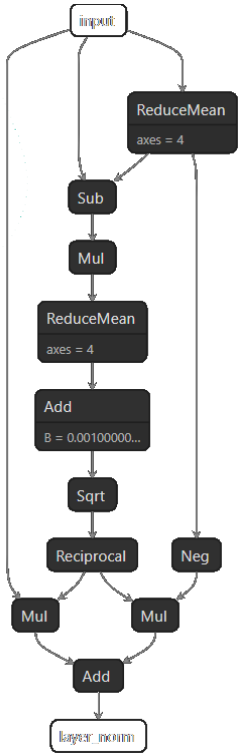
Automatically simplify ONNX graphs

Example with LayerNorm fusion




```
# LayerNormalization from Torch ONNX
fuse_layernorm(model,
    ".#0~*>ReduceMean#1~*>Sub#2~*>Pow
#3->ReduceMean#4[axis]~*>Add#5-
>Sqrt#6~*>Div#6;"
    ".#0~*>Sub#2~*>Div#6;"
    "Pow#3<*~Producer;"
    "Add#5<*~Producer[epsilon];"
    "Div#6~*>(Mul#7~*>Add#8)?;"
    "(Mul#7<*~Producer[scale])?;"
    "(Add#8<*~Producer[bias])?;")
```

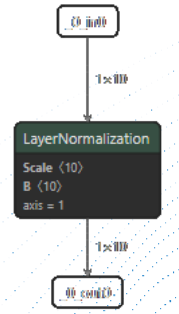
or



Exported from **PyTorch**

Exported from **Keras**

How to handle complex, non canonical sub-graphs?



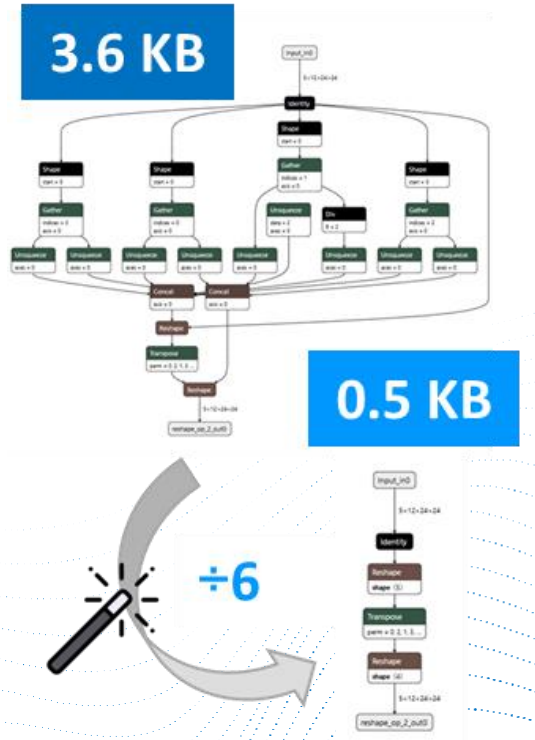
Automatically simplify ONNX graphs

Graph simplification through pattern matching:

- Fuse LayerNorm / Gelu w.r.t to Pytorch / Keras variants
- Constant/Shape folding
- MatMul + Add → Gemm
- 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 with the CLI
- **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	X	✓	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)

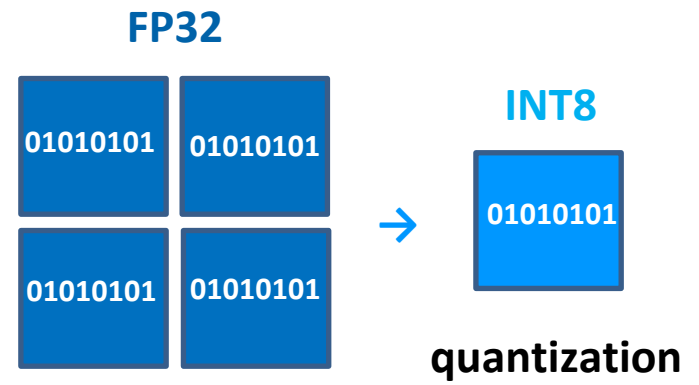
Model	Execution Time	Throughput	vs Best
original	0.552 ms	1812.0 inf/s	+0.007 ms (+1.3%)
onnx_cleaner	0.545 ms	1835.4 inf/s	BEST
onnxsim	0.550 ms	1817.1 inf/s	+0.005 ms (+1.0%)
onnxslim	0.554 ms	1806.1 inf/s	+0.009 ms (+1.6%)

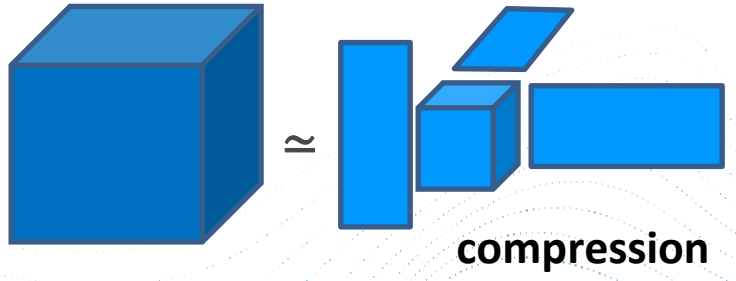
Neural networks optimization

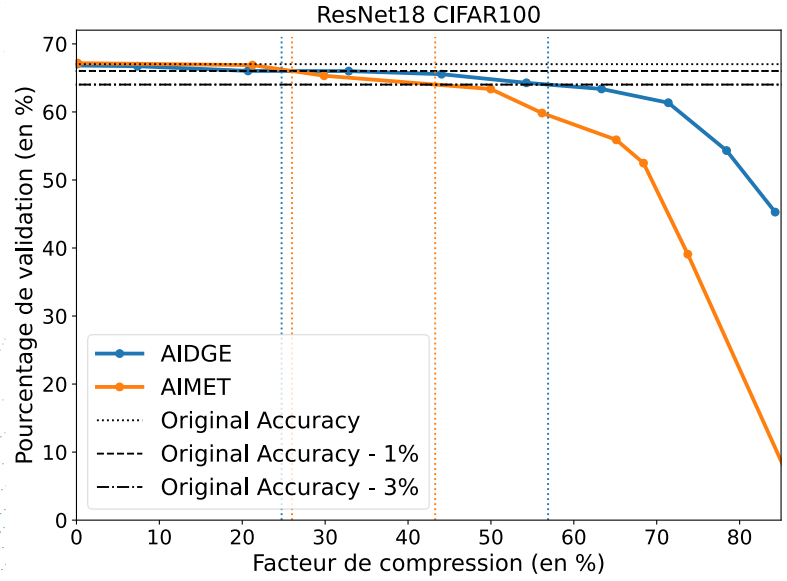
- **Quantization:** convert parameters to lower precision data types
- **Compression:** decomposes resource-intensive layers into multiple smaller operations

- **Smart & versatile optimization**
  - **Hardware agnostic:** deployment across diverse hardware targets
  - **Precision control**: choose the right accuracy/performance trade-off
  - **Complementary**: quantization and compression can be seamlessly combined

- **Breakthrough performance** (outperforming AIMET)
  - Achieve up to **60% model compression**
  - Experience up to **30% faster inference**







Complete framework with innovations

aidge

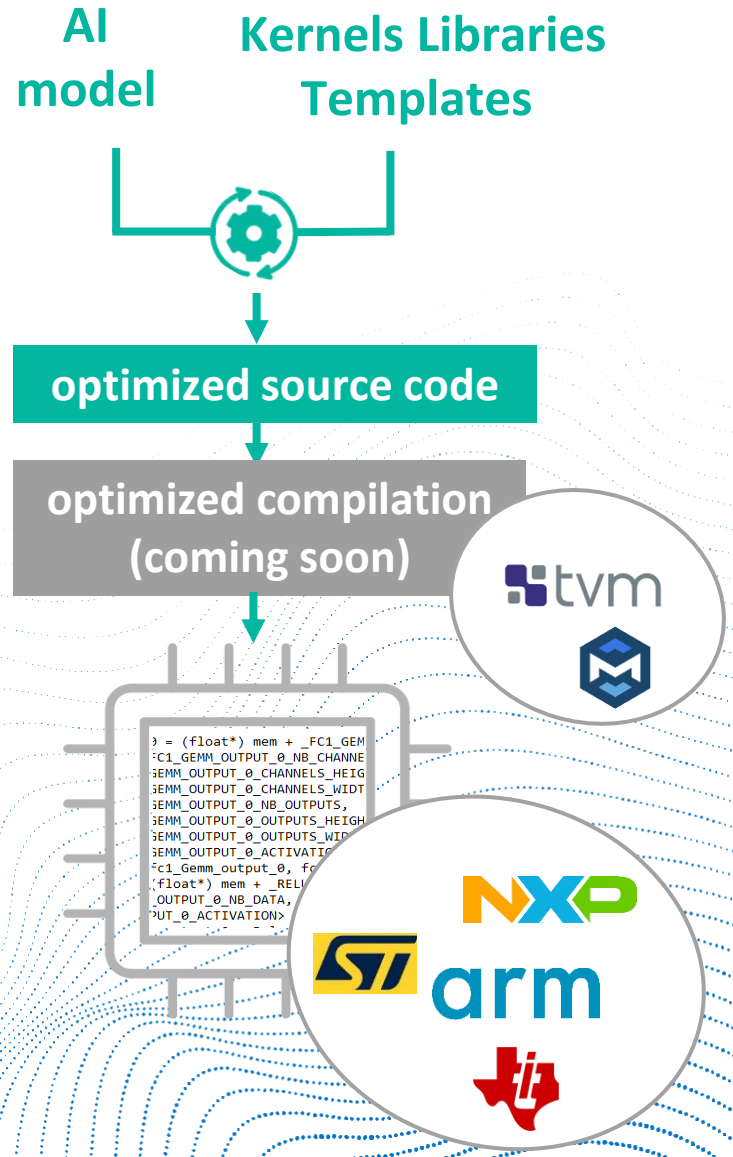
Easily embed optimized AI

- **Universal compatibility**

Export your optimized AI models to ONNX ensuring broad interfacing with countless SDKs and tools.

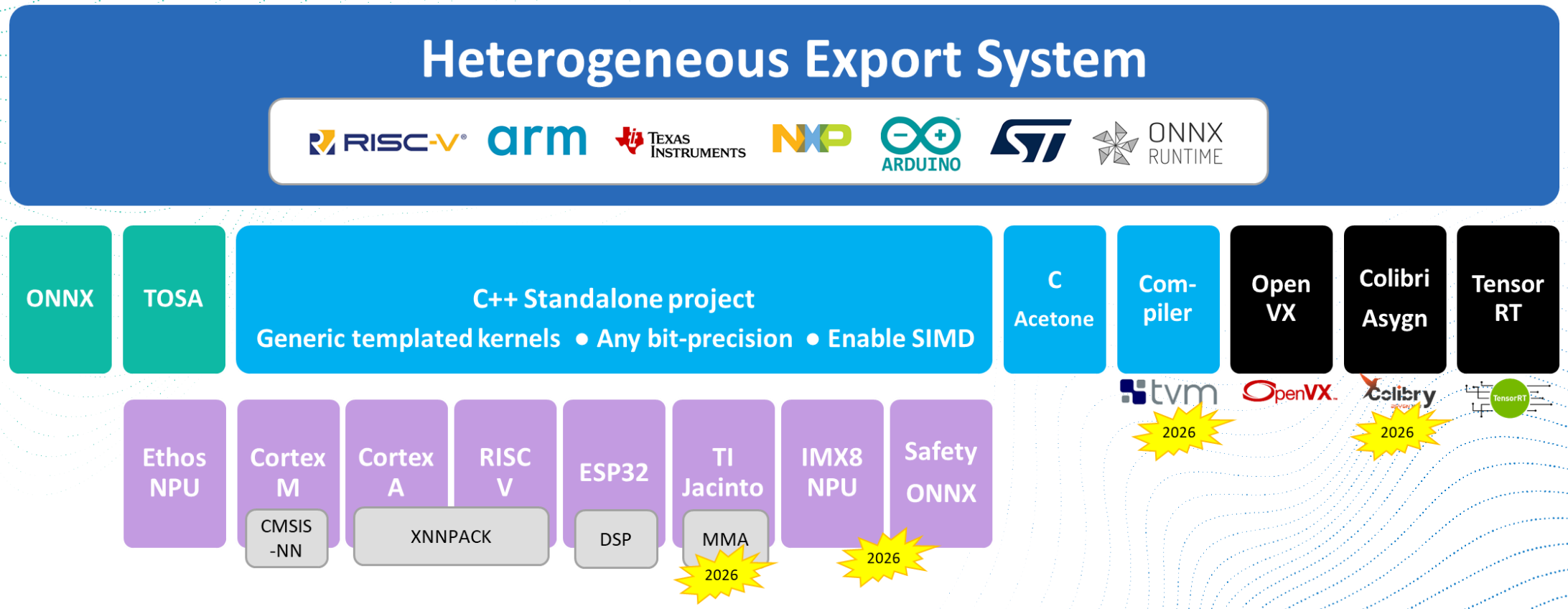
- **Intelligent and optimized code generation:**
  - **Transparent code engine** (C++/C) that automatically generates high-quality and standalone source code
  - **Multi-target C++ reference export**
  - **Specializations** for ARM, Texas Instrument SoC and ESP32,…
  - Support for third-forparty low-level libraries such as XNNPack and CMSIS-NN

Control the orchestration and the memory optimization



aidge

Easily embed optimized AI

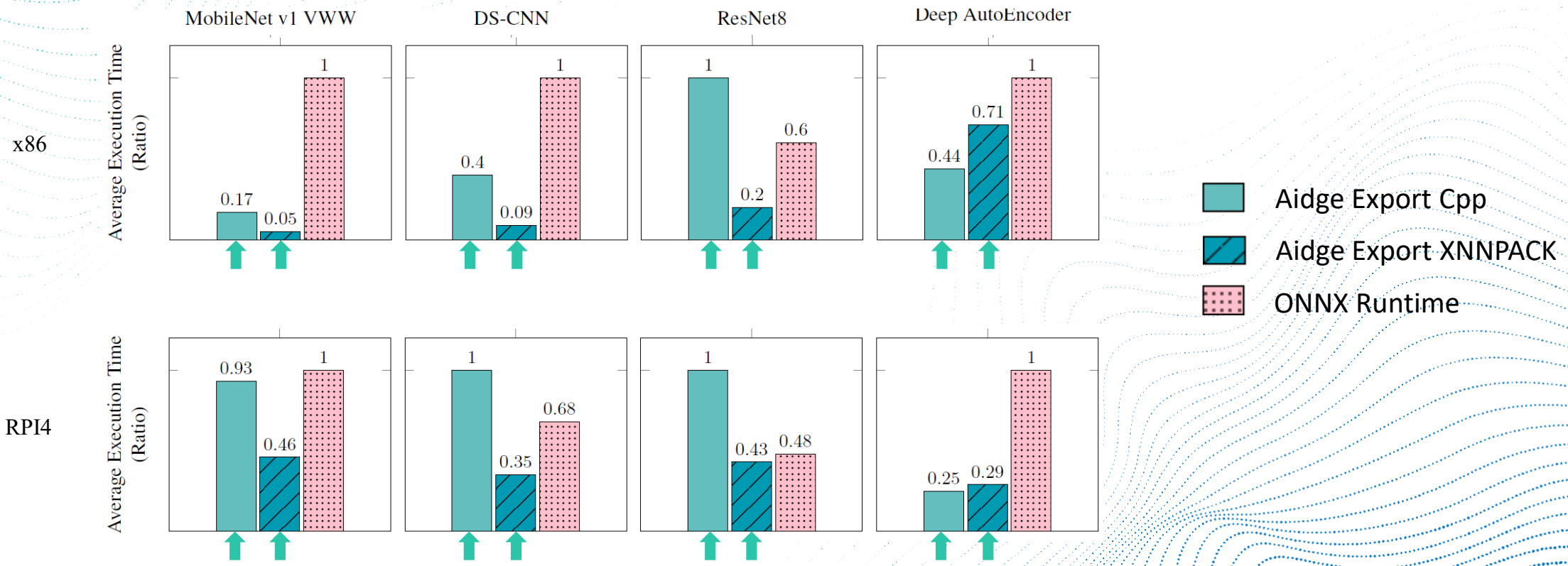


Built by

cea · ONERA THE FRENCH AEROSPACE LAB · Inria · MountAIn · IRT SAINT EXUPÉRY · asygn · Micelia

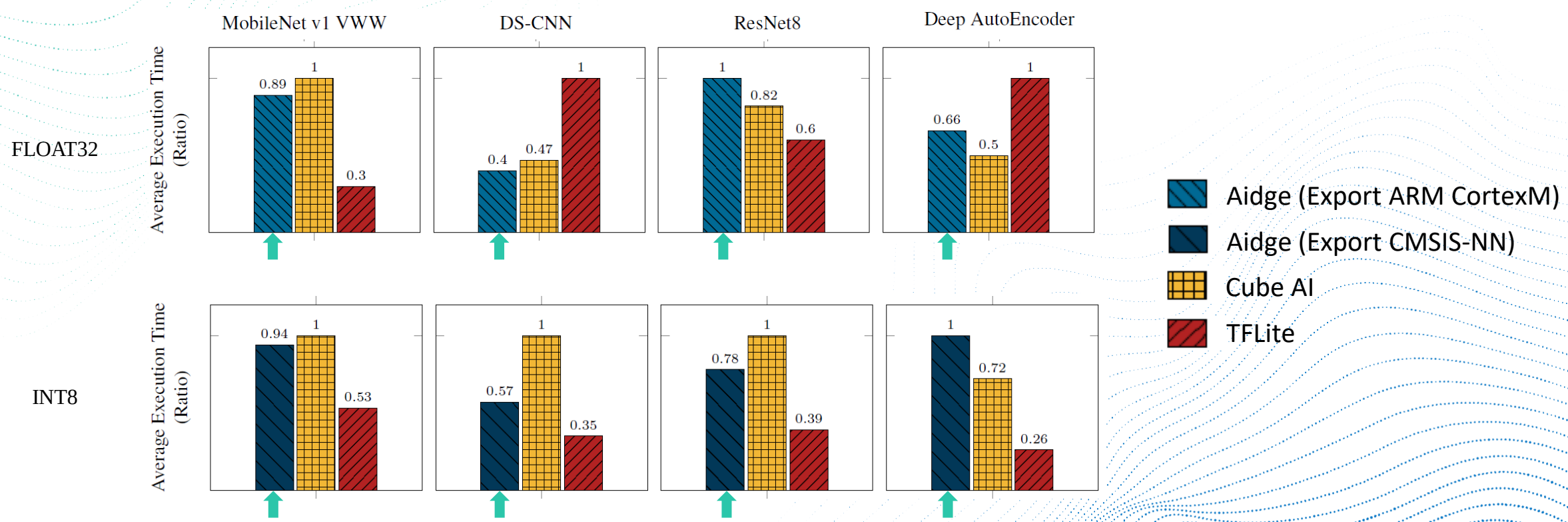
Benchmark – Inference of export cpp

- **Export Cpp is general purpose**, compatible with any compiler compliant with C++14 standard
- We can use third parties library such as **XNNPack** to **speed up inference time**!
- No **compromise in transparency.** Memory mapping, scheduling and operator fusion are decided by the user.

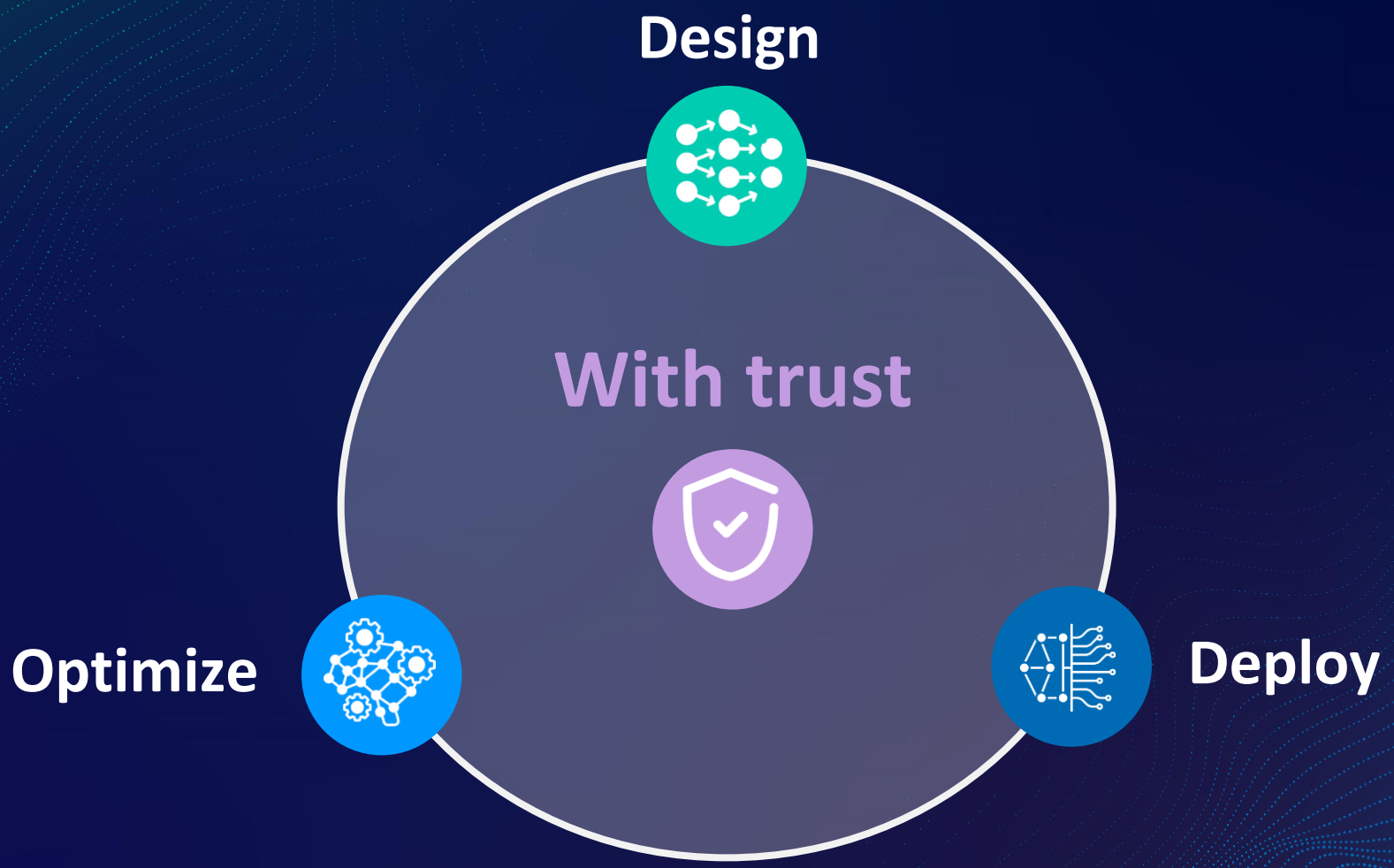


Benchmark – Inference on STM32H743ZI

On hardware targets with dedicated accelerator, Aidge is comparable to vendor solution on state of the art deep neural network architectures



Complete framework with innovations



aidge

Building trustworthy embedded AI

- ☑ **Audit** with open and verifiable source code
- ☑ **Strengthen model resilience** through robust learning approach
- ☑ Test neural network inference under **hardware fault conditions**
- ☑ **Pioneer** in the adoption of the new Safety ONNX format (being the reference implementation of the standard)
- ☑ **Meet strict aeronautical standards** using the ACETONE plugin

Reference implementation of the Safety ONNX

ONNX has not been developed with safety in mind and needs to be clarified, verified, completed,…

Objectives of the S-ONNX Working Group :

- Provision of an informal and formal specification of ONNX operators
- Tracability between informal and formal specification
- Formal verification of formal specification
- Generation of a **reference implementation of operators**

Let a be the concatenation axis and $d_{k,a}$ **(T2)** the dimension of the X_k input tensor k along the axis a.

Let s_k be the cumulative offset along axis before input X_k as:

$$\text{T2: } s_k = \sum_{j=0}^{k-1} d_{j,a}$$

Let i_a be the global index along dimension a, and let i'_a be the corresponding local index within a local tensor X_k. This relationship can be defined as follows:

$$\text{T3: } i'_a = i_a - s_k$$

If the global index i_a satisfies the condition:

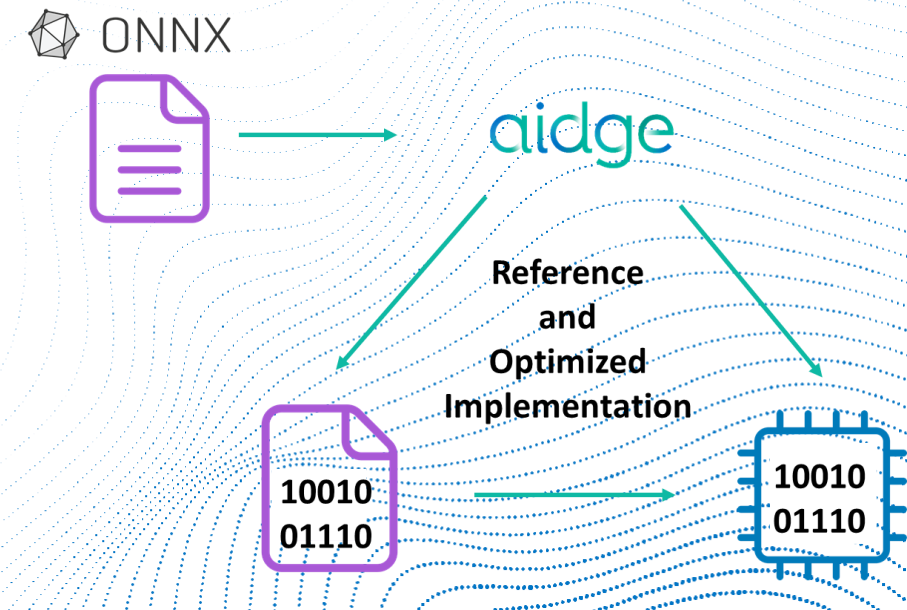
$$\text{T4: } s_k \leq i_a < s_k + d_{k,a}$$

then the relationship holds:

$$\text{T5: } \forall i_0, \ldots, i_{r-1}.\ Y[i_0, \ldots, i_{r-1}] = X_k[i_0, \ldots, i'_a, \ldots, i_{r-1}]$$

With i_0 and i_{r-1} are the indices which access respectively the first and last dimensions of a **r-dimensional** tensor. $i_0, \ldots, i_{r-1}$ represent a set of indices that uniquely identify an element within an **r-dimensional** tensor.

```
(** T3: defining the local index i' for a i (global index) given **)
(*
  seq_D_axis: seq_D_axis is the sequence of all the d_k,axis for each k tensor
  i_axis: value on the axis along which the concatenation is performed
  k: Current index in sequence seq_D_axis. Must be 0 at the first call of the
  function
  s_k: Current sum of the previous dimensions in seq_D_axis. Must be 0 at the
  first call of the function
*)
let rec rec_find_k_and_i_prime (seq_D_axis: seq int) (i_axis: int) (k: int)
(s_k: int) : (int, int)
 variant { length seq_D_axis - k }  (* Termination measure *)
=
  if i_axis < s_k + seq_D_axis[k] then (* Inequality (T5) to define to in which
  k tensor the local index i' is defined *)
    (k, (i_axis - s_k)) (* Definition (T5) by keeping the upper part of the
    inequality: i' = i - s_k *)
  else
    (* The global index i is superior to s_k + seq_D_axis[k] so the k tensor to
    define the local index i' is not the current one but next one k+1. Update
    the offset (s_k) by adding the length of the tensor we just checked
    (current tensor). *)
    rec_find_k_and_i_prime seq_D_axis i_axis (k + 1) (s_k + seq_D_axis[k])
```

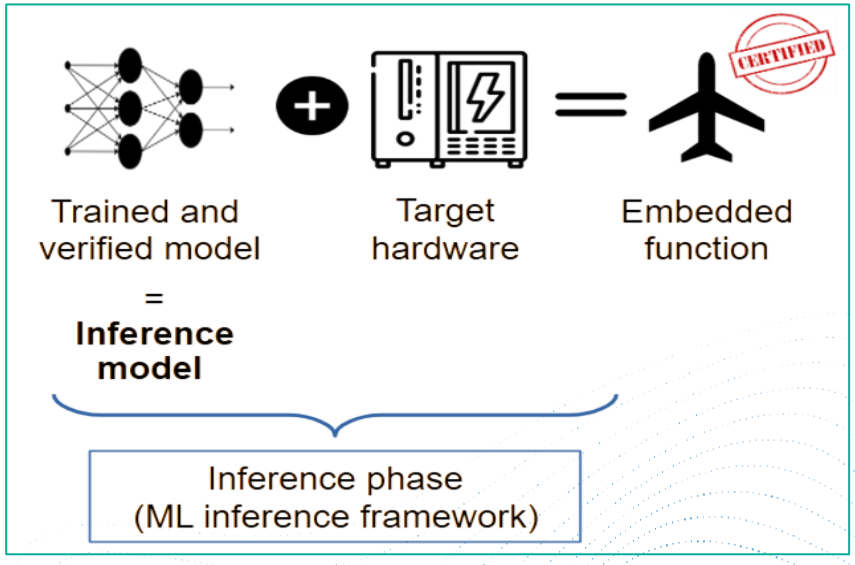


Safety and Certification : ACETONE module

By Iryna de Albuquerque Silva, Benjamin Lesage and Filipo Perotto

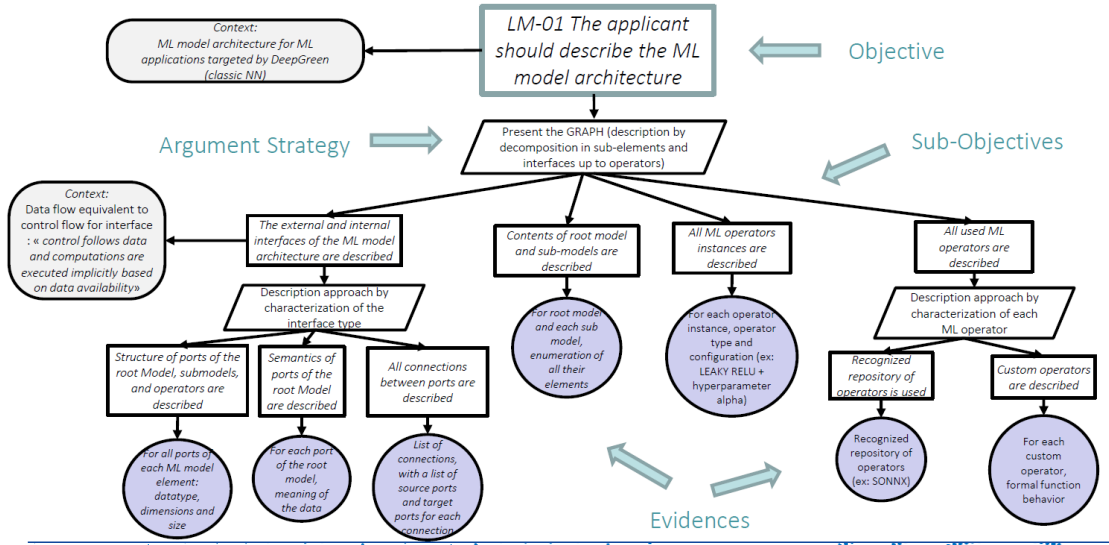
Context

- Avionics systems are subject to strict certification procedures (DO-178C, ARP)
- **Existing ML frameworks do not comply** with derived certification objectives
- How to make neural network inference both efficient and certifiable in avionics systems?



Objectives

- **Semantics preservation** → same behavior on target as in training environment
- **Requirements traceability** → mapping between the inference model description, the C code and the associated binary file
- **Timing predictability** → determine tight worst-case execution time (WCET) upper bounds for the inference
- **Efficient resources usage** → examine memory usage and average execution times



Safety and Certification : ACETONE module

By Iryna de Albuquerque Silva, Benjamin Lesage and Filipo Perotto

ACAS-Xu use case

- Study of the **airborne collision avoidance** system approximation based on fully-connected neural networks
- **Compared frameworks:** Aidge, Keras2C, Static microTVM

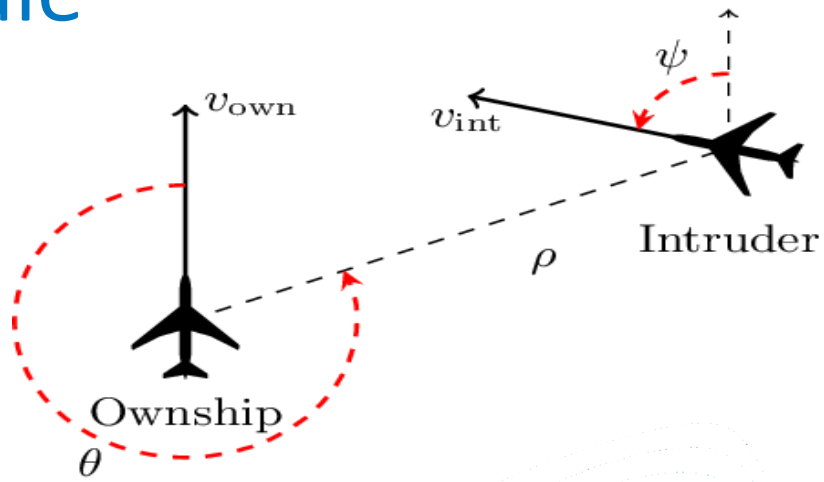


Figure: Ownship, intruder and sensor measurements of the ACAS-Xu system.

Evaluation criteria and results

- **Semantic preservation:** numerical difference between outputs of learning and inference frameworks for a given input

  → **Errors ~10^{-6} (FP32) for all frameworks**

- **Timing predictability:** static analysis with OTAWA tool

  → **WCET bounds derivable; comparable for ACETONE and Static microTVM, overly pessimistic for Keras2C**

- **Efficient resources usage:** average of the observed times of 50 executions on an ARM Cortex-A15 processor

  → **Equivalent to Static microTVM, ~2× faster on average than Keras2C**

aidge
Benchmarking solution

Goal

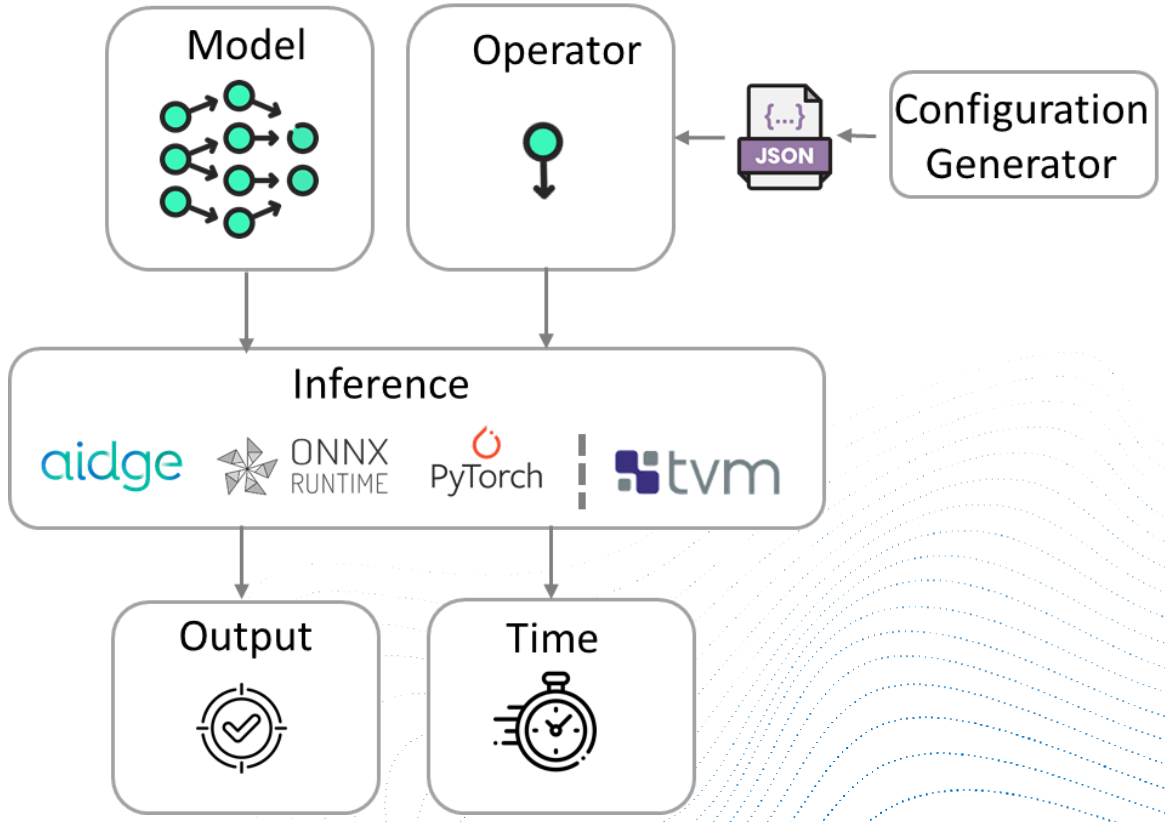
- Compare output results across backends
- Measure and compare inference time

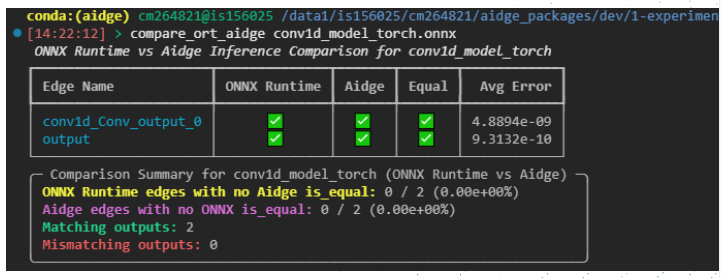
Multi-backend support

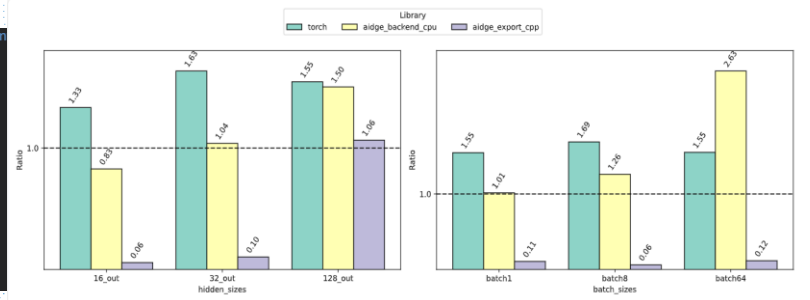
- Aidge backend CPU or CUDA or any export
- External runtimes: ONNX-RT, PyTorch, TVM (coming soon)

Benchmarking scope

- Full model benchmarking
- Operator-level benchmarking using customizable configurations







Active developments and collaborations

+60 submitters

+10 organizations

+300 Commits /month

v0.9 version

20cent ikucher bhalimi obichler fabricea mnewson gkubler pineapple wboussella jeromeh raphaelmillet lrakotoarivony diegob marwaabd louislerbourg sylvainbataille alalloyer macario vbaudelet mszczep yberkat flebert farnez

oantoni jsimatic nvrlosemyself idealbuquerque clementgf julienl nthm bobot alemesle thibaultallenet jgirardsatabin lucaslopez vlorrain axelfarr cguillon silvanosky hleborgne operrin mick94 na25 lsoulier cmoineau noamzerah alicebatte hrouis

Norms

EUROCAE afnor ONNX

+30 industrials partners

THALES Building a future we can all trust, infineon, AIRBUS, KALRAY, ALSTOM, MBDA MISSILE SYSTEMS, Valeo, ArcelorMittal, NX NanoXplore, ST, EDF, SAFRAN, KNDS, ...

+10 academic partners

SAL SILICON AUSTRIA LABS, Fraunhofer, cea, Inria, ONERA THE FRENCH AEROSPACE LAB, cnrs, université PARIS-SACLAY, ...

Real world impact 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 25% the peak memory**
- Deployed on Audiomoth / Silicon Labs

Heat Pump Monitoring

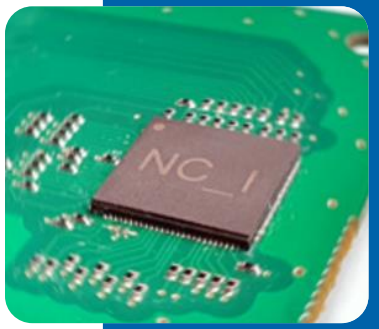
- **Lightweight prediction algorithm** for adaptive heat pump control
- Up to **40% energy saving** for collective housing
- Deployed on STM32

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



Key challenges ahead

① Ease support of heterogeneous architectures

② Integration of compilation toolchain (TVM / MLIR)

③ Enable update of model over time (continual and federated learning)

Edge AI revolution

Evolution of embedded computing for mobility

Centralized cloud

Distributed cloud

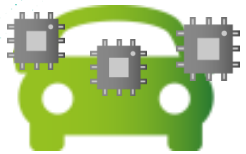
Update of new applications

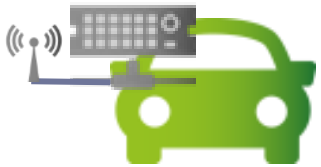
Cooperation

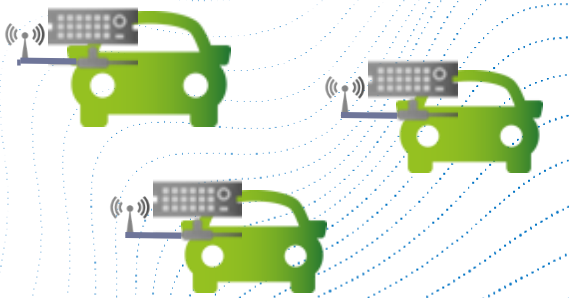
Electronic Control Units

Centralized architecture

Distributed servers







Sense & Compute

Software-Defined Vehicle

Decentralized & Embedded Intelligence

Today

Tomorrow

Future

Join us

Gitlab

Code repository, issues, and discussions

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

Documentation

User guide, API and +20 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

Follow us

Join us

Do not wait on major foreign third-party provider to hope **to beat time-to-market**...
Become a contributor and take part of the roadmap and development process **to suit your needs**