ONNX quantization representation formats

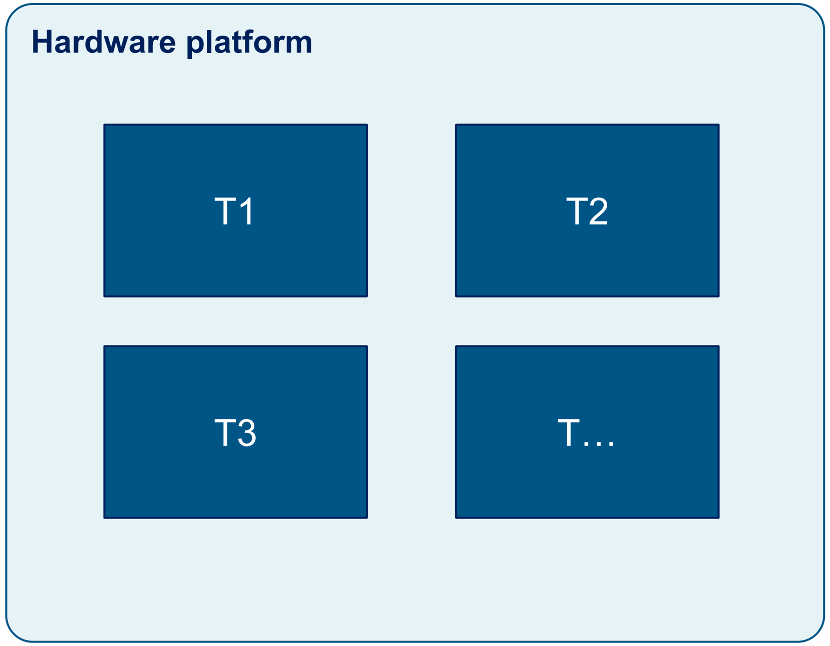
Alex DIGONNET - 05/07/2025

AIRBUS

Introduction

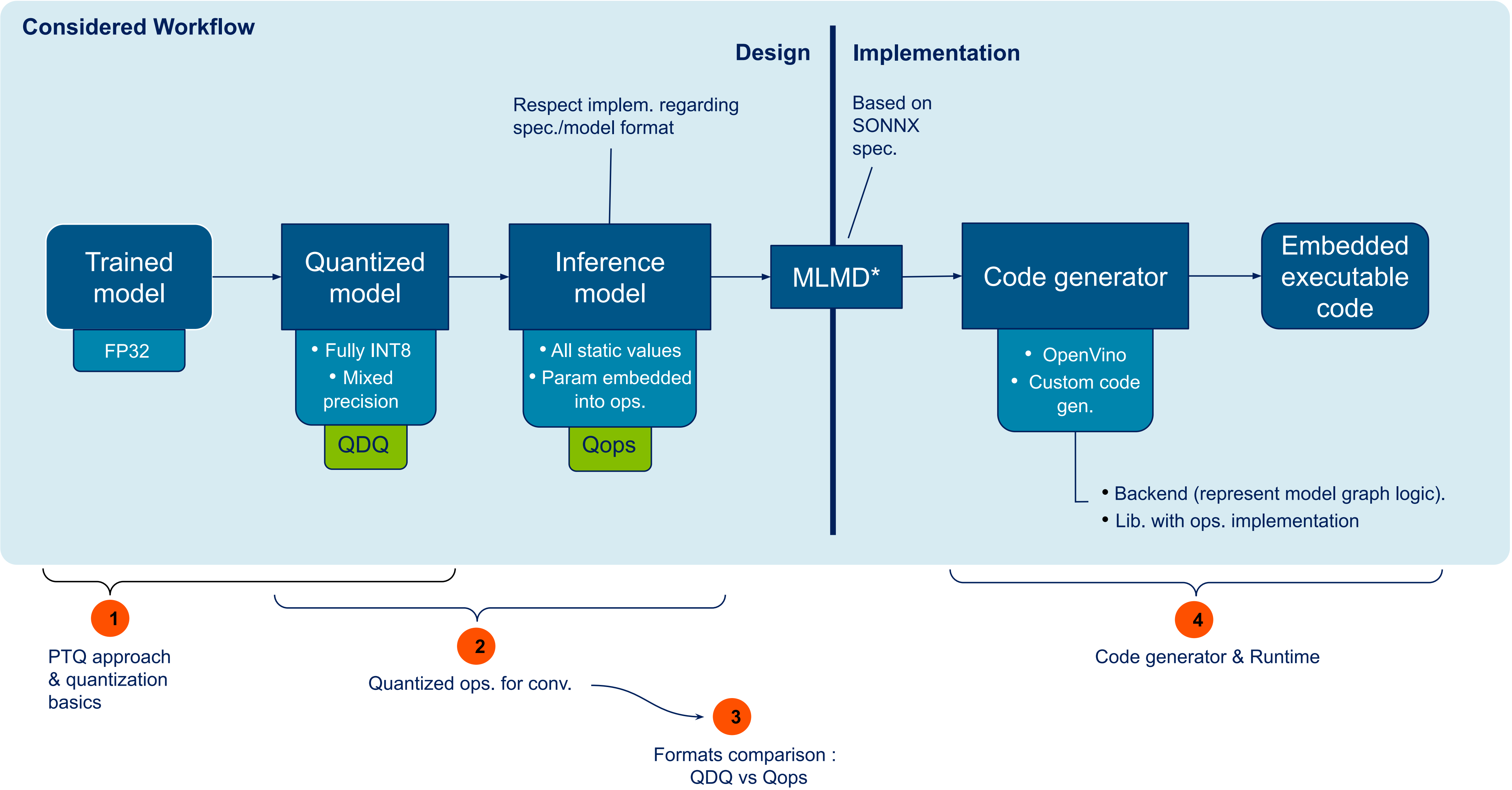
Why do we want to quantize our models at Airbus?

- ➔ Improving inference speed.
- ➔ Reducing energy consumption.
- ➔ Facilitating the deployment of multiple tasks.



AIRBUS

Context & Presentation Objectives



***MLMD - Machine Learning Model Descriptor (ARP 6983)**

AIRBUS

1) Quantization Basics: Parameters & Approaches

1. Mapping Parameters

- Quantization maps FP32 values to INT8/UINT8.
- This requires two key parameters per tensor:
  - **scale:** The step size between quantized values (FP32).
  - **zero-point:** The INT8/UINT8 value corresponding to FP32 zero.
- ONNX Needs: A way to calculate and carry/store these scale and zero-point values

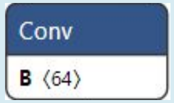
Fig 1: Quantization ONNX operators

2. Calculation Approaches: Dynamic vs. Static

- *Static Quantization*
  - scale/zero-point calculated offline.
  - Requires a "**calibration dataset**" to determine typical value ranges (min/max).
  - Parameters are **fixed and stored within the ONNX model**.
  - Pros: Faster inference (parameters pre-computed); predictable bounds.
  - Cons: Needs representative calibration data; accuracy depends on calibration quality.

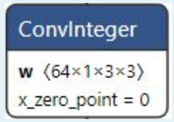
- *Dynamic Quantization*
  - **scale/zero-point calculated on-the-fly** during inference.
  - Based on runtime min/max values (often for activations only).
  - Pros: No calibration data needed; potentially more accurate if value ranges vary significantly.
  - Cons: Higher inference cost; unpredictable parameter bounds.

2) Quantized Conv Operators



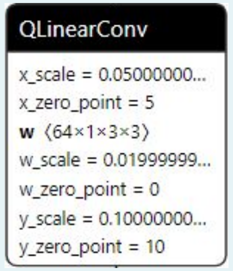
Not INT8 compatible; it only supports floating-point or double data types.

Can’t carry INT8 parameters



Performs INT8 matrix multiply-add operations exclusively.

Handles mixed-precision models: e.g. external bias computation



Fully compatible with integer models.

Performs matrix multiply-add operations (INT8) and bias computation (INT32).

AIRBUS

3) ONNX Formats: QDQ (Quantize/Dequantize)

Tensor-oriented (QDQ) mechanism:

- Generic format that inserts pairs of QuantizeLinear (Q) / DequantizeLinear (DQ) nodes around original FP32 operators.
- Simulates quantization/dequantization processes within the graph.
- Evaluates functional precision losses.

Why QDQ is Common:

- **Compatibility:** Supported across many frameworks (PyTorch, OpenVino, ONNX).
- **Decoupling:** Separate quantization representation (Q/DQ) from core quantization computations (original ops).

Limitation :

- **Implicit:** Exact quantization process isn't explicitly defined by nodes.
- **Execution:** An Execution Provider (e.g., OpenVINO) is needed to interpret this format for running the model in INT8. Not efficient for inference.

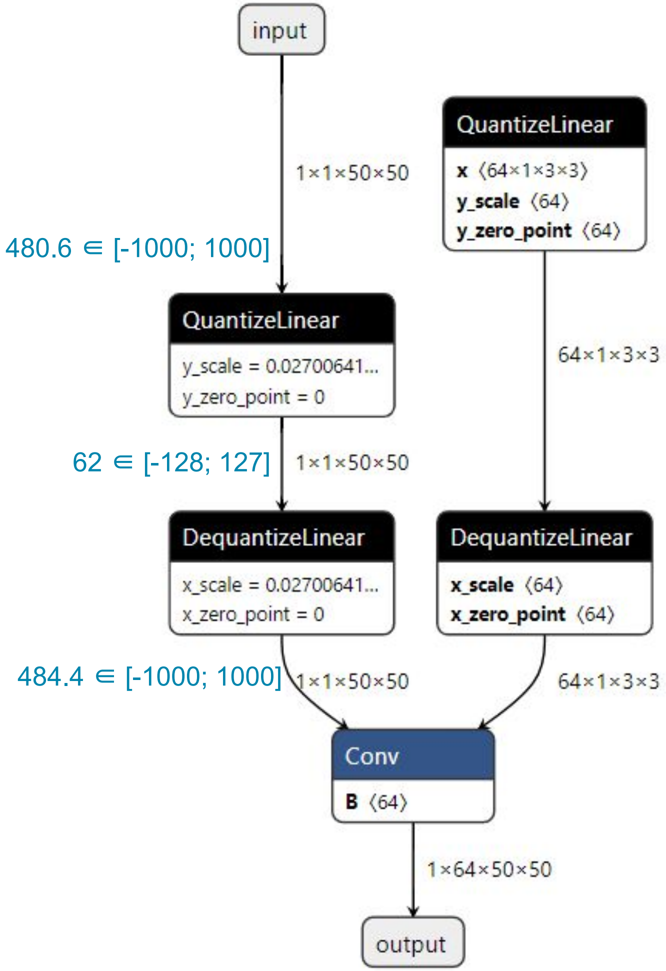


Fig 2 : QDQ format

3) ONNX Formats: QOps (Operator-Oriented)

Qops Mechanism:

- Explicitly defines the quantization process with operators.
- Uses dedicated low-precision operators (e.g. QLinearConv, ConvInteger).
- Format available when using ONNX quantizer.

Advantages:

- **Explicit Representation:** Direct representation and clearly defines a quantization process
- **Native Integer ops:** Operators carry and directly handle low-precision data/parameters.

Limitations:

- **Limited compatibility:** Direct integer execution is not possible.
- **Graph transformation:** Requires specific tools to convert models QDQ into QOps format.
- **Operator Coverage:** Not all ONNX operators have corresponding QOps version defined or implemented.

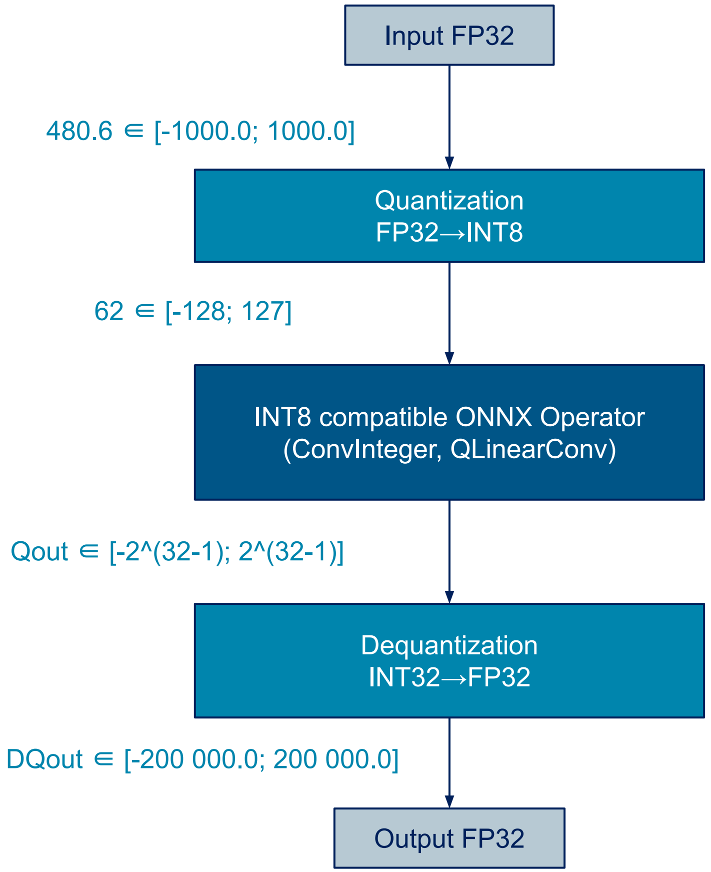


Fig 3 : Qoperators format

3) Qops-Based Quantization Structure : ConvInteger

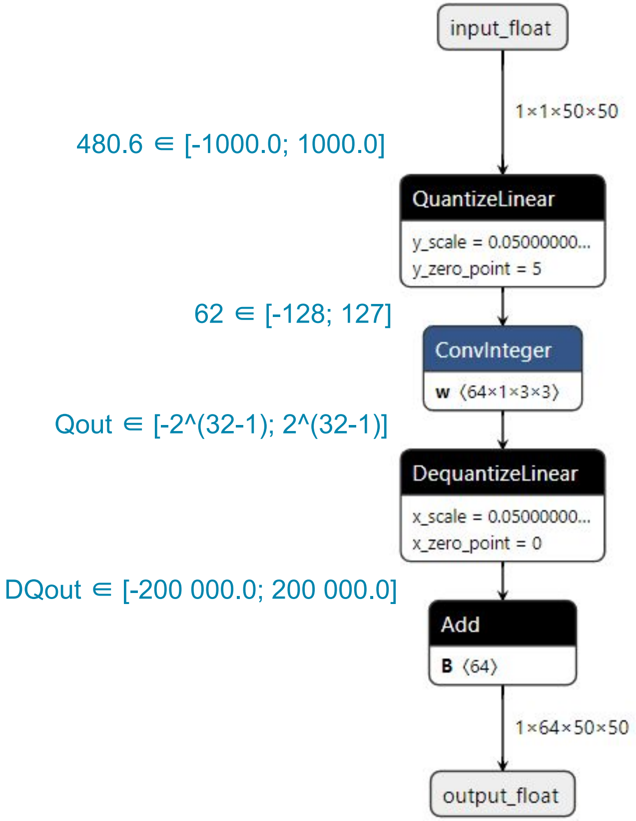


Fig 4 : ConvInteger-based representation

Advantages

- **Custom Quantization Flow:** Full control over the quantization process (scales, zero-points, and mathematical logic).
- **Mixed precision compatible:** Enables bias computation in floating-point precision.
- **Flexible:** Works directly on int8/uint8 tensors with external scale management.

Limitations

- **More Complex Graph:** Less readable in terms of end-to-end quantization compared to QLinearConv.

Additional Information

- ConvInteger was introduced for dynamic quantization in ONNX, but it seems that there are no constraints on using it in static mode.

3) Qops-Based Quantization Structure : QLinearConv

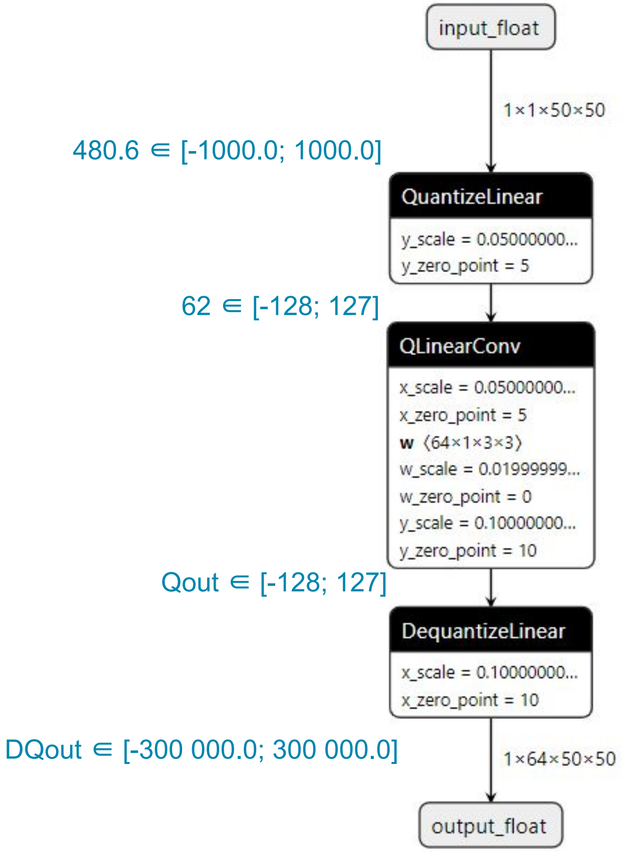


Fig 5 : QLinearConv-based representation

Advantages

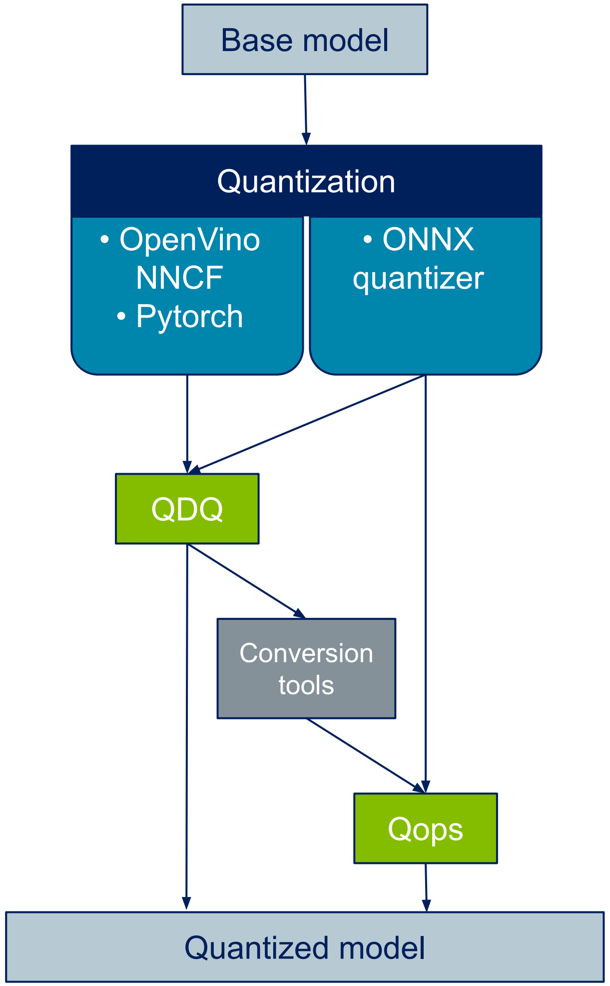
- **Simplified Graph:** Fully encapsulates quantization logic (scale/zero-point) within a single operator.
- **Fully-integer models :** Easier to interpret.

Limitations

- **Less Customization Flexibility:** Offers a fixed quantization flow.
- **Integer-Only Bias:** Bias must be computed from quantized inputs/weights and must be int32.

.

3) QDQ to QOps conversion



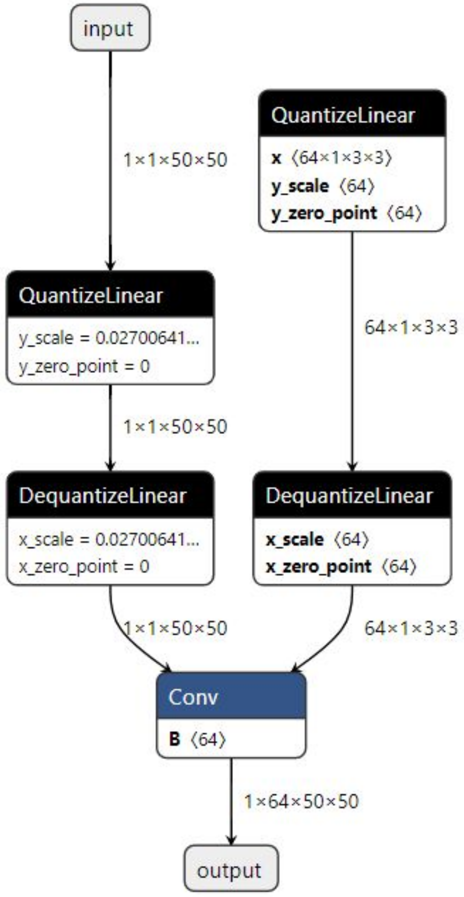


Fig 6 : QDQ format

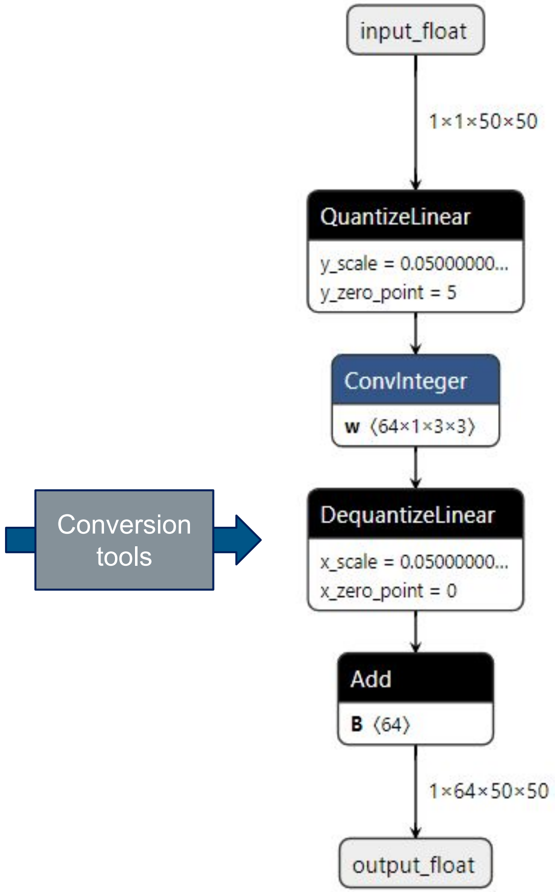


Fig 7 : Qops format

Minimize transformations between the model graph and its implementation.

AIRBUS

4) OpenVino Compatibility: QDQ Models & Optimization

Model Preparation: ONNX to OpenVINO IR

Convert the Quantize-Dequantize (QDQ) ONNX model to OpenVINO Intermediate Representation (IR) format (.xml and .bin files).

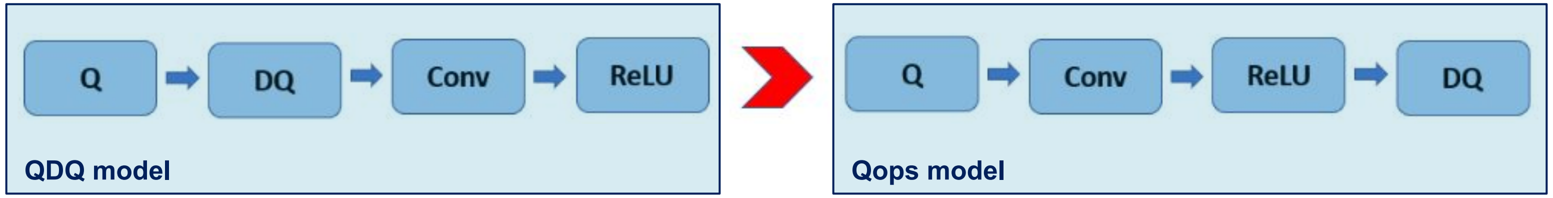


Fig 8 : LPT transformation QDQ to “Qops” format

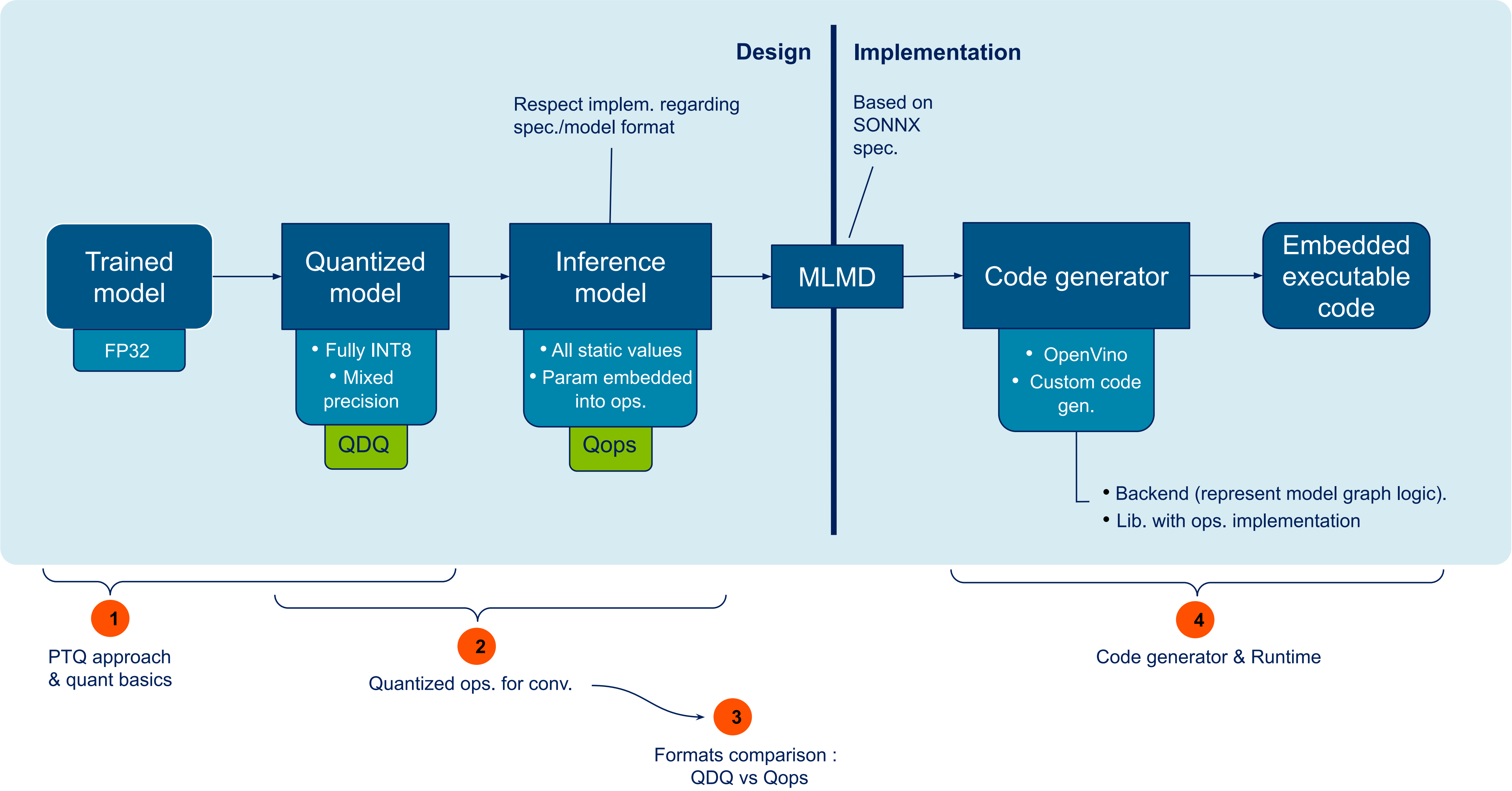
Low Precision Transformations (LPT)

- **Goal:** To transform the model graph into an optimized low-precision representation (typically INT8/UINT8).
- **Mechanism:** LPT is a component within OpenVINO that analyzes the model before inference.

How does LPT Interprets QDQ Models at Runtime ?

- **QDQ Pattern Recognition:** LPT specifically looks for the "fake quantization" patterns introduced by QuantizeLinear / DequantizeLinear (QDQ) node pairs in the model graph.
- **Output:** LPT applies transformation rules to operators surrounded by QDQ patterns, converting these graph sections into actual low-precision operators (e.g. INT8 Convolution) for efficient execution by the OpenVINO runtime.

Considered Workflow



AIRBUS