QuantLab: a Modular Framework for Training and Deploying Mixed-Precision NNs
Matteo Spallanzani¹, Georg Rutishauser¹, Moritz Scherer¹, Alessio Burrello², Francesco Conti² and Luca Benini¹,²
¹ETH Zürich (Switzerland), ²Università di Bologna (Italy)

Deploying ML Solutions at the Edge

Research in tinyML algorithms is continuously proposing new training algorithms for quantised neural networks (QNNs), while hardware designers have introduced architectural support for sub-byte and mixed-type integer arithmetic.

QuantLab aims to help developers creating the most effective quantised neural networks (QNNs) using the best training algorithms and allowing for mixed-precision policies, and to facilitate their deployment on tinyML devices.

QuantLib & QuantLab

QuantLib is based on PyTorch, and consists of two components:
• QuantLib, the quantisation library;
• QuantLab, the experiment management front-end.

QuantLib:
• supports sub-byte and mixed-precision quantisers;
• supports several quantisation algorithms;
• can be used as a plug-in for PyTorch projects.

QuantLab:
• enables easy comparisons between different data sets and network architectures;
• minimises the duplication of ML system components;
• facilitates the generation of statistically solid results.

Quantisers

Quantisation-aware training (QAT) algorithms embed the integer ranges (true-quantised, TQ) to be used at execution time into fake-quantised (FQ) ranges.

Quantisers map floating-point (FP) ranges to target FQ ranges.

| PRECISION: n ∈ ℤ, n > 1 |
| OFFSET: e ∈ ℤ, e > 0 |
| SCALE: z ∈ ℤ |

![Diagram](image)

**Figure 1:** the computational graph of a quantiser σ.

**Figure 2:** a portion of the computational graph of an FP network (a), of a canonicalised FP network (b); of an FQ network (c).

**Fake-quantised networks** use FP operators (weights and features) to mimic quantisation at training time.

QuantLib supports the programmatic transformation of FP networks (Figure 2b) into FQ ones:
• graph canonicalisation: for instance, replacing non-modular operators with modular API, or folding the bias of linear operations into the following batch-normalisations (Figure 2b);
• point-wise replacement of PyTorch nn.Module objects with FQ counterparts (Figure 2c);
• ε-harmonisation of additions and concatenations (Figure 3).

QuantLib supports several QAT algorithms using dedicated FQ nn.Module objects:
• STE: straight-through estimator;
• ANA: additive noise annealing;
• INQ: incremental network quantisation;
• PACT: parametric clipping activation; and
• TQT: trained quantisation thresholds.

![Diagram](image)

**QuantLab:** Experiment Management

Users develop mixed-precision QNNs via JSON configuration files (what vs. how).

Users can define and execute factorial experimental designs simply by scripting how to patch configuration files.

QuantLab supports automatic cross-validation.

Fake-to-True Conversion

To enable deployment on tinyML devices, FQ networks must be rewritten in terms of backend-supported integer operations while preserving functionality. The resulting programs are true-quantised networks.

QuantLib supports a composable and transformable collection of transformations to rewrite FQ networks graphs into TQ ones:
• ε-propagation: annotate each FP array with the corresponding scale factor (Figure 4a);
• arithmetic folding: use elementary arithmetic properties (e.g., distributive, commutative) to expose TQ arrays (Figure 4b);
• requantisation: approximate the remaining FP operations using the requantisation property (Figure 4c).

The output of the fake-to-true conversion process is an ONNX file, annotated with the precision of each operand.

![Diagram](image)

**Example Use Case:** MobileNets on PULP

Two experiments:
• compare different QAT algorithms in conjunction with homogeneous quantisation policy, and verify correctness of the fake-to-true conversion (Table 1);
• compare homogeneous to mixed-precision policies to fit a tinyML device (Table 2).

![Table](image)

**Table 1:** accuracy of an 8-bit MobileNetV2 network, trained with different QAT algorithms; note that in both cases, fake-to-true conversion is almost lossless (some small errors might be introduced by arithmetic folding and requantisation due to the imperfect correspondence between FP and integer arithmetic).

![Table](image)

**Table 2:** performance of a MobileNetV1 network, trained using different QAT algorithms and two different quantisation policies. Measurements were taken using the GVSIS simulator, emulating a PULP system integrating the sub-word XpuNN arithmetic extensions; code was generated using the DORY tool. Note that the mixed-precision policy (Table 2) removes the need of accessing writing to and reading features from off-chip RAM (L3 memory).

![Table](image)

**Table 3:** the MobileNetV1 mixed-precision quantisation policy that avoids accessing off-chip RAM. The layers that are not reported use 8-bit weights and features.

Code & Contacts

QuantLib and QuantLab are open-sourced on GitHub!
• QuantLib: [https://github.com/pulp-platform/quantlib](https://github.com/pulp-platform/quantlib)
• QuantLab: [https://github.com/pulp-platform/quantlab](https://github.com/pulp-platform/quantlab)

They have been developed in the scope of the parallel ultra-low-power (PULP) project.

[spmatteo@iis.ee.ethz.ch](mailto:spmatteo@iis.ee.ethz.ch)