QuantLab: a Modular Framework for Training and Deploying Mixed-Precision NNs

Matteo Spallanzani[1], Georg Rutishauser[1], Moritz Scherer[1], Alessio Burrello[2], Francesco Conti[2] and Luca Benini[1,2]

[1]ETH Zürich (Switzerland), [2]Università 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

QuantLab 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: $n \in \mathbb{N}, n > 1$
OFFSET: $z \in \mathbb{Z}$
SCALE: $\varepsilon \in \mathbb{R}, \varepsilon > 0$

TQ RANGE: $\{z, z+1, \dots, z+n-1\}$
FQ RANGE: $\varepsilon\,\{z, z+1, \dots, z+n-1\}$

$$\sigma : \mathbb{R} \to \varepsilon\,\{z, z+1, \dots, z+n-1\}$$
$$x \mapsto \varepsilon\,\lfloor clip(x/\varepsilon, z, z+n-1+\varepsilon/4) \rfloor$$

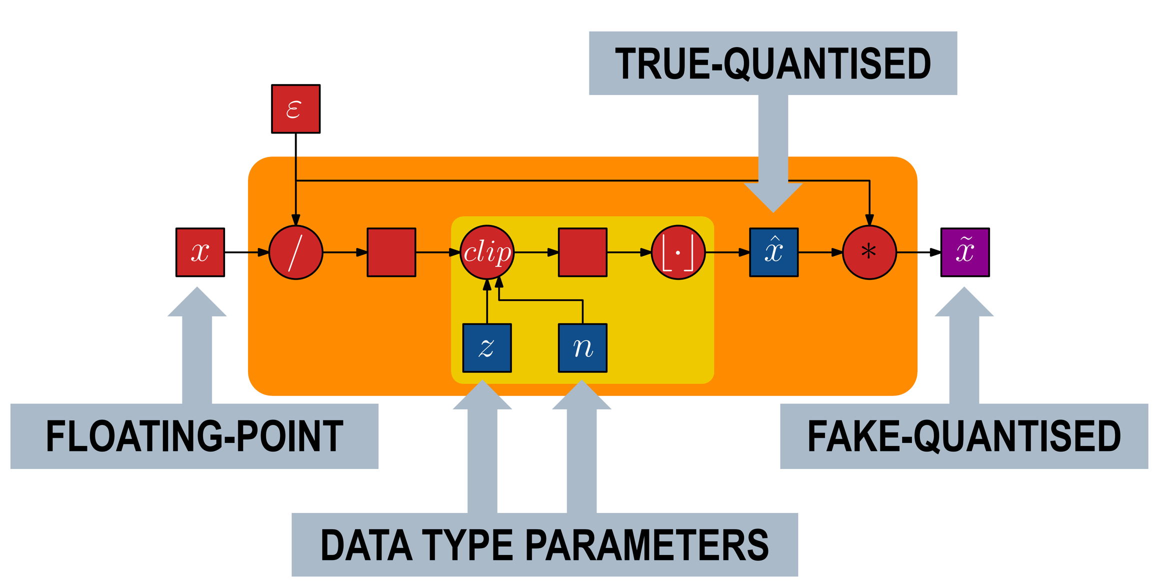


Figure 1: the computational graph of a quantiser σ.

Float-to-Fake Conversion

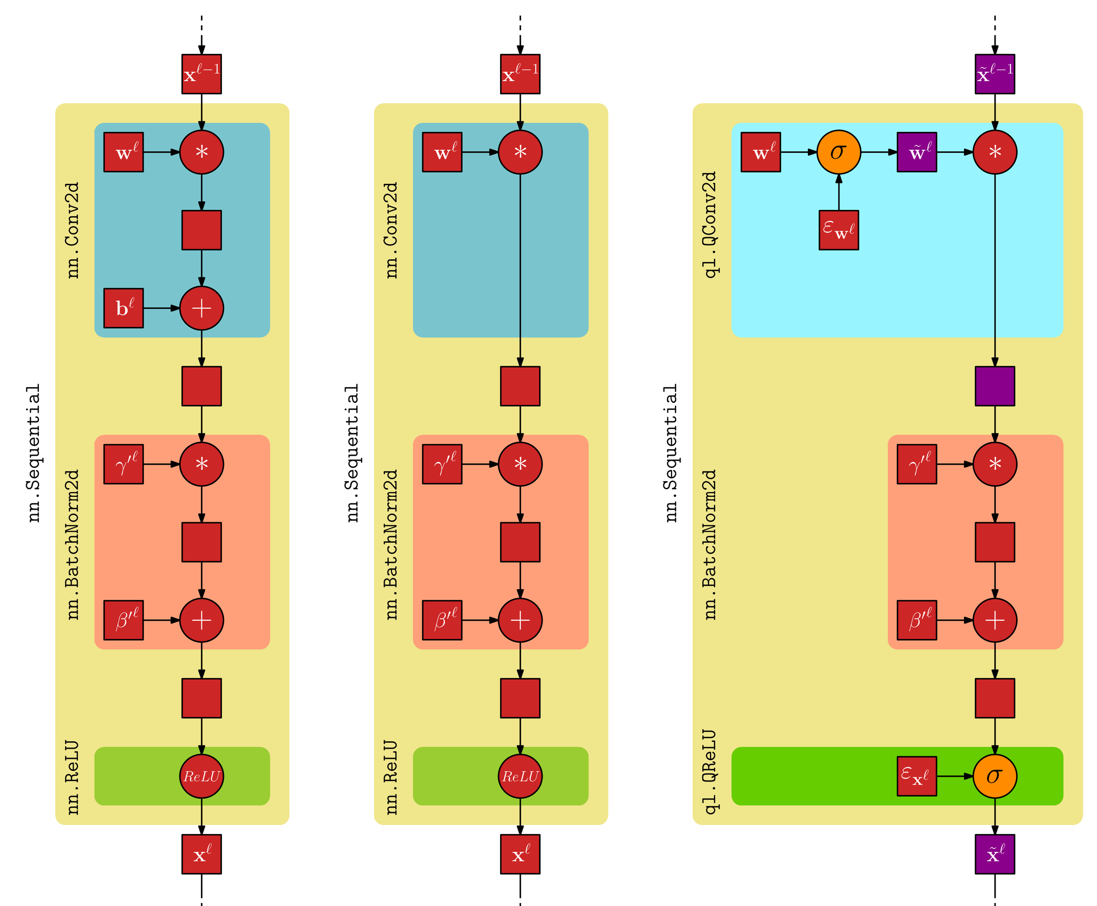


Figure 2: a portion of the computational graph of an FP network (a); of a canonicalised FP network (b); of an FQ network (c).

Fake-quantised networks use FQ operands (weights and features) to mimic quantisation at training time.

QuantLib supports the **programmatic transformation of FP networks** (Figure 2a) into FQ ones:

- **graph canonicalization**; for instance, replacing non-modular 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-harmonization** 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**: parametrised clipping activation;
- **TQT**: trained quantisation thresholds.

ε-Harmonisation

Consider two FQ arrays:

$$\tilde{x}_A = \varepsilon_A \hat{x}_A,\ \tilde{x}_B = \varepsilon_B \hat{x}_B.$$

Under which conditions can we write their sum as an FQ array

$$\tilde{x}_A + \tilde{x}_B = \varepsilon_C \hat{x}_C?$$

$$\varepsilon_A = \varepsilon_B$$

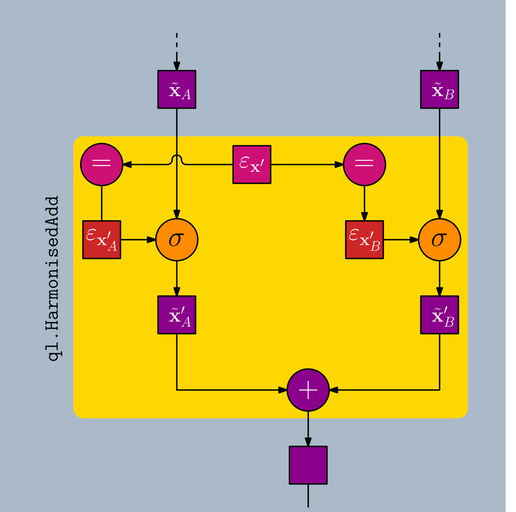


Figure 3: detail of the ε-harmonised computational graph of a ResNet-like network (merging of the residual into the identity).

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 an **composable and extensible collection of transformations** to rewrite FQ networks graphs into TQ ones:

- **ε -propagation**: annotate each FQ 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)

$$\lim_{D \to +\infty} \lfloor 2^D x \rfloor / 2^D = x\ .$$

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

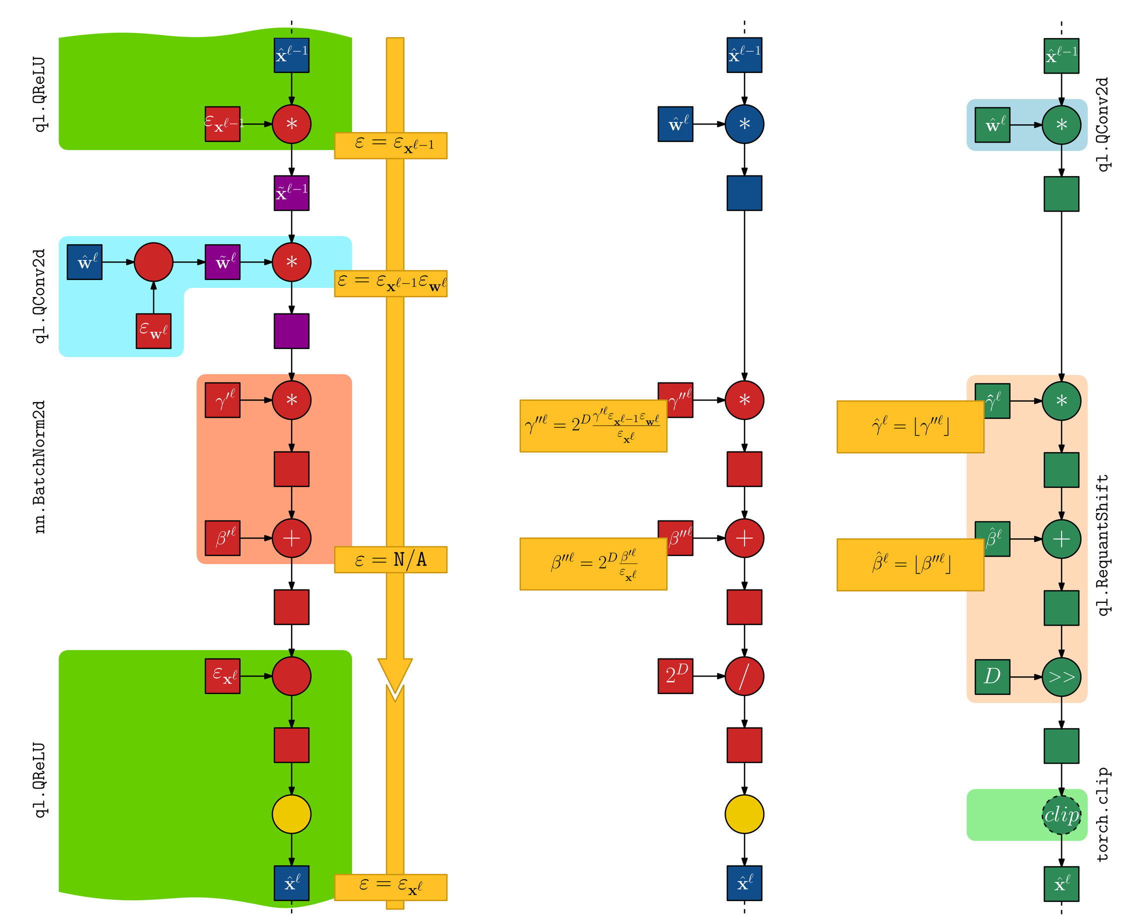
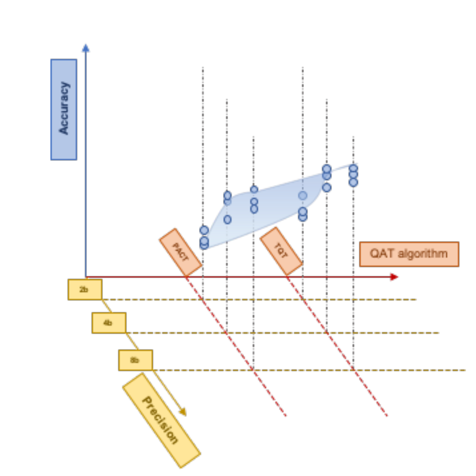


Figure 4: a portion of the computational graph of an FQ network after ε-propagation (a); of a partially integerised FQ network (b); of a fully integerised (TQ) network (c).

QuantLab: Experiment Management

Users **describe 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**.



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).

	PACT	TQT
Accuracy (FQ)	71.4%	71.4%
Accuracy (TQ)	71.3%	71.4%

Table 1: accuracy of an 8-bit MobileNetV2 network, trained with two 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).

	8-bit	Mixed	Relative
Accuracy (PACT)	69.2%	65.9%	-4.8%
Accuracy (TQT)	69.4%	67.0%	-3.5%
Latency [ms]	705.50	557.10	-21.0%
Energy (total) [mJ]	38.17	30.11	-21.1%
Energy (math) [mJ]	35.98	29.41	-18.3%
Energy (L3) [mJ]	2.19	0.70	-68.0%
L3 accesses ($\hat{w}$) [#]	2568161	2266880	-11.7%
L3 accesses ($\hat{x}$) [#]	4515840	0	-100.0%

Table 2: performance of a obileNetV1 network, trained using two different QAT algorithms and two different quantisation policies. Measurements were taken using the GVSoC simulator, emulating a PULP system integrating the sub-word XpulpNN arithmetic extensions; code was generated using the DORY tool. Note that the mixed-precision policy (Table 3) removes the need of accessing writing to and readimg features from off-chip RAM (L3 memory).

	φ_1	φ_2	φ_3	φ_4	φ_5	φ_6	φ_7	φ_8	...	φ_26	φ_27
$\hat{w}$	8	4	4	4	8	4	4	4	...	8	4
$\hat{x}$	4	4	2	8	4	4	4	8	...	2	8

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

QuantLab and QuantLib are **open-sourced on GitHub**!

- QuantLib: https://github.com/pulp-platform/quantlib/
- QuantLab: https://github.com/pulp-platform/quantlab

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

e-mail: spmatteo@iis.ee.ethz.ch