

From Layers to Gradients at 100 Qubits: The TQML Quantum Machine Learning Platform

Maniraman Periyasamy¹, Viacheslav Kuzmin¹, Asel Sagingalieva¹, Alexey Melnikov¹

¹Terra Quantum AG, Kornhausstrasse 25, 9000 St. Gallen, Switzerland

Motivation. Quantum machine learning (QML) spans a broad ansatz design space with very different simulation demands: hardware-efficient and NISQ-style variational circuits, data re-uploading models with learnable frequencies, depth-infused and exponential-feature encodings, quantum and hybrid recurrent architectures, matrix-product (tensor-network) circuits, and, increasingly, symmetry- and particle-number-preserving ansätze such as fixed-Hamming-weight sectors, Givens/RBS networks, and equivariant geometric constructions. General-purpose simulators collapse this diversity onto a dense 2^n statevector, capping differentiable training near 25 to 30 qubits and forcing gate-level bookkeeping that does not compose with autograd. TQML inverts this: it exposes the whole spectrum as composable ML layers over a simulator that adapts its state representation to the ansatz.

The TQML platform. TQML is a set of packages sharing one namespace, `tqml.*`, that close a full design loop inside ordinary PyTorch and JAX: suggest an architecture, analyse and refine it, train, benchmark, and feed the findings back. `tqnet` and `tqsim` are the live core; the surrounding modules are detailed below.

tqnet: layers as modules. Every layer is at once a PyTorch `nn.Module` and a JAX/Equinox module from one framework-agnostic circuit recipe, so the front-ends are numerically identical and chosen by a single import. A layer is declared by its design axes, not built gate by gate: encoding, entangling template, rotation axes, measurement mode, depth, and differentiation method are constructor arguments, from which `tqnet` assembles the parameterised circuit and returns its measurement as a batched, device-aware, differentiable tensor operation; backprop, parameter-shift, and adjoint all run on native `tqsim` kernels. That single abstraction spans variational, hardware-efficient, re-uploading, recurrent, and tensor-network forms through to structured, symmetry-preserving (Hamming-weight) constructions.

tqsim: a simulator built for QML. `tqsim` decouples what is computed from how it executes, pairing a backend (Torch with CPU/GPU/MPS autograd, JAX with JIT and `vmap`, or a C++ path with fast-Walsh-Hadamard and Kronecker- R_z kernels) with a state representation (dense 2^n , matrix-product state, or the k -sector subspace). The subspace engine is the scaling lever: it evolves excitation-preserving circuits in the fixed-Hamming-weight sector, storing the k excited indices in a $[\binom{n}{k}, k]$ layout with $\mathcal{O}(k)$ combinadic rank/unrank, so R_z layers, Givens/RBS single-excitation brick-layers, and Z/ZZ observables cost $\mathcal{O}(k\binom{n}{k})$ rather than $\mathcal{O}(n2^n)$, all under autograd. A deferred intermediate representation feeds a two-pass optimiser (peephole rewrites plus same-kind layer fusion) and an auto-router that chooses between layered kernels (bit-trick R_z , Hadamard-sandwich R_x , permutation-gather CNOT) and a whole-circuit tensor contraction, with per-circuit and per-machine calibration fitting the hot path.

The design loop. `tqcirc` runs trainability and expressivity diagnostics: quantum Fisher-information matrices and their eigenspectra, barren-plateau detection via gradient-variance scaling with log-slope fits over qubit count and depth, Fourier-spectrum analysis of re-uploading models, and ZX-calculus simplification. `tqoptim` supplies optimisers for variational landscapes, including quantum-natural-gradient updates preconditioned by the `tqcirc` Fisher metric, SPSA, and barren-plateau-aware rules, drop-in beside `torch.optim`. `tqtune` maps data and prior features to a candidate architecture and training hyperparameters, optionally via an in-house suggestion service, and refines it against `tqcirc` reports so untrainable configurations are rejected before training. `tqbench` measures trained-model quality with wall-clock, memory, and scaling across backends and sizes, feeding design suggestions back to `tqtune`.

Results. Everything reported was measured on the platform. In the weight-3 subspace, `tqsim` computes a full forward pass and gradient for circuits of up to **100 qubits**, a 161,700-dimensional space, in a few seconds on a laptop CPU, far beyond any 2^{100} statevector. Gate-level correctness is validated across every backend and state representation by an extensive automated test suite. End to end, a Hamming-weight-preserving quantum convolutional network trained on MNIST reaches **86.6% test accuracy**.

Significance. The upshot is one stack that carries a model from proposal to trained result without leaving PyTorch, at scales where dense simulators cannot return a gradient.