



DistributedEstimator: Distributed training of quantum neural networks via circuit cutting[☆]

Prabhjot Singh^{ID*}, Adel N. Toosi, Rajkumar Buyya

Quantum Cloud Computing and Distributed Systems (qCLOUDS) Lab, School of Computing and Information Systems, The University of Melbourne, Melbourne, VIC, 3010, Australia

ARTICLE INFO

Keywords:

Quantum neural networks
Circuit cutting
Distributed quantum computing
Quantum machine learning
Variational quantum algorithms
Circuit knitting
Distributed training

ABSTRACT

Circuit cutting decomposes a large quantum circuit into smaller subcircuits that are executed independently; the original circuit's expectation values are then recovered by classically combining the measured subcircuit outcomes. While prior work characterises cutting overhead in terms of subcircuit counts and sampling complexity, its end-to-end impact on iterative, estimator-driven training pipelines remains insufficiently measured from a systems perspective. We propose *DistributedEstimator*, a cut-aware estimator execution pipeline that treats circuit cutting as a staged distributed workload. Each estimator query is instrumented across four phases: partitioning, subexperiment generation, parallel execution, and classical reconstruction. Using logged runtime traces and learning outcomes on two binary classification workloads (Iris and MNIST), we quantify cutting overheads, scaling limits, and sensitivity to injected stragglers, and evaluate whether accuracy and robustness are preserved under matched training budgets. Our measurements reveal that reconstruction constitutes a dominant fraction of per-query time—reaching a median of 53% and a 95th percentile of 58% at three cuts—thereby bounding achievable speed-up under increased parallelism. Despite these overheads, test accuracy is fully preserved on Iris and maintained without systematic degradation on MNIST across all evaluated cut configurations. Robustness under Gaussian noise and FGSM perturbations is similarly preserved, with several cut configurations exhibiting comparable or improved robustness relative to the uncut baseline. The exponential growth of subexperiment counts with each additional cut ($\mathcal{O}(9^c)$ for CNOT-based decomposition) represents a fundamental computational barrier that limits practical experimentation to small qubit counts with current methods. These results establish that practical scaling of circuit cutting for learning workloads requires reducing and overlapping reconstruction, designing scheduling policies for barrier-dominated critical paths, and developing computationally efficient reconstruction strategies for larger qubit counts.

1. Introduction

Hybrid quantum–classical workloads are increasingly shaped by systems constraints rather than purely algorithmic ones: the circuits and training loops of practical interest frequently exceed the width, depth, or reliability envelope of near-term devices, while still requiring repeated expectation estimation in an interactive optimisation loop [1–3]. In this regime, execution is dominated by *estimator-heavy* pipelines that combine many short quantum jobs with non-trivial classical orchestration, aggregation, and optimiser feedback [4,5]. As a result, the end-to-end performance of quantum machine learning (QML) and variational training is governed not only by circuit latency, but also by how effectively we schedule, parallelise, and reconstruct large collections of sub-tasks under heterogeneous runtimes [6–8].

Circuit cutting provides a principled mechanism to execute circuits that do not fit a target backend by decomposing them into smaller subcircuits that can be executed independently, followed by classical reconstruction of the desired quantity [9–12]. This transforms a single circuit evaluation into a staged distributed workload: a fan-out of subcircuit executions, followed by a global reduction step that reconstructs the target expectation value (and, in learning settings, supports gradient estimation and optimiser updates). The methodological benefits of cutting are clear. The systems implications are less well characterised. Existing work typically reports overhead in terms of subcircuit counts, sampling complexity, or asymptotic reconstruction cost [10,12,13], which is informative for feasibility but does not directly answer the operational question that governs scaling: *where does wall-clock time go when we actually run the pipeline in parallel?*

[☆] This article is part of a Special issue entitled: 'Adv. Quantum Vol. IV' published in Future Generation Computer Systems.

* Corresponding author.

E-mail address: prabhjot.singh.1@student.unimelb.edu.au (P. Singh).

1.1. Motivation and problem statement

We identify three practical limitations that motivate our study.

P1: Overhead attribution is under-specified. Cutting introduces compilation, materialisation, dispatch, and reconstruction costs that are often treated as secondary or summarised coarsely. Without stage-level instrumentation, it is difficult to determine whether slowdowns are intrinsic to cutting or artefacts of the runtime.

P2: Reconstruction can dominate the critical path. Cutting increases execution parallelism but concentrates work in a reconstruction stage. Analogous reduction phases bound scalability in distributed computing even when upstream work parallelises well [6,8].

P3: Training dynamics, accuracy, and robustness must be assessed end-to-end. In learning pipelines, the optimiser observes finite-sample estimators with potentially different noise profiles; small changes can alter optimisation trajectories [2–4]. Both predictive accuracy and robustness [14,15] must be verified under the same training protocol.

These are systems problems: they concern staging, scheduling, aggregation, and the interaction between distributed execution and iterative optimisation.

1.2. Research questions

We structure the paper around the following research questions (RQs), designed to be answered by measured runtime logs and training traces rather than by asymptotic arguments:

1. *RQ1 (Cutting overhead):* What additional end-to-end overheads are introduced by circuit cutting compared to an uncut execution baseline, and how do these overheads scale with the degree of parallelism?
2. *RQ2 (Scalability bottlenecks):* Which pipeline stages bound scaling under parallel execution, and under what conditions does classical reconstruction become the dominant contributor to wall-clock time?
3. *RQ3 (Scheduling policy and staggering):* How do execution policies that stagger or otherwise reshape subcircuit dispatch affect throughput, straggler sensitivity, and overall time-to-solution?
4. *RQ4 (Accuracy and training behaviour):* Does cutting preserve task accuracy under identical training protocols, and can the induced estimator and scheduling characteristics measurably influence training stability (e.g., convergence behaviour) relative to the uncut baseline?
5. *RQ5 (Robustness):* How does circuit cutting affect robustness metrics under the same evaluation regime, and are observed changes attributable to estimator effects rather than to changes in model definition?

Each RQ is framed to admit an unambiguous operationalisation: we measure stage-level durations, derive scaling trends and bottleneck shares, and evaluate accuracy and robustness from recorded training and evaluation traces.

1.3. Approach overview

To answer these questions, we design an estimator-based execution pipeline that makes cutting explicit as a staged distributed workload: (i) generate a cut plan and the associated subcircuits [10,12], (ii) execute subcircuits as independent tasks on a worker pool, (iii) reconstruct target estimates via a dedicated aggregation phase, and (iv) feed reconstructed estimates into an iterative training loop. Each stage is instrumented to produce time-stamped records, enabling stage-level attribution under varying parallelism and scheduling policies. We do not assume proportional speed-ups; instead, we quantify where the critical path shifts as we scale out, and couple runtime measurements with training logs to ensure performance improvements do not come at the cost of model quality (RQ4–RQ5).

The pipeline integrates with the Qiskit machine learning stack: *EstimatorQNN* [16] wraps the parameterised circuit into a differentiable layer, and *TorchConnector* bridges it to PyTorch. When cutting is enabled, *DistributedEstimator* intercepts each estimator query, decomposes it into subexperiments via *qiskit-addon-cutting* [17], dispatches them, reconstructs the expectation value, and returns it transparently. Gradients computed via the parameter-shift rule [4] generate additional estimator queries, each independently decomposed and reconstructed through the same pipeline, ensuring the training loop remains unchanged regardless of whether cutting is enabled.

1.4. Contributions

Our main contributions are as follows:

- *A measurement-driven formulation of circuit cutting as a distributed staged pipeline*, with instrumentation that attributes end-to-end time across cutting, subcircuit execution, reconstruction, and training, and with explicit integration into the QML training loop via *EstimatorQNN* and *TorchConnector*.
- *An empirical scalability analysis grounded in stage-level logs*, enabling bottleneck identification and explaining when additional parallelism does not translate into reduced wall-clock time due to aggregation and coordination effects.
- *A study of scheduling and straggler sensitivity for cutting workloads*, with a framework supporting eager, batched, and staggered dispatch, evaluated through measured throughput and tail behaviour under straggler injection rather than assumed ideal parallelism.
- *An end-to-end assessment of learning outcomes*, reporting accuracy and robustness under consistent training protocols to distinguish performance gains from changes in optimisation behaviour.
- *A characterisation of computational scalability barriers*, documenting the exponential growth of reconstruction overhead with cut count and establishing the computational constraints that limit practical experimentation to small qubit counts with current methods.

Together, these contributions establish a systems-level understanding of performance trade-offs from circuit cutting in estimator-heavy training pipelines and provide a reproducible basis for reasoning about overheads, bottlenecks, and outcome preservation.

The rest of the paper is organised as follows. Section 2 presents the technical background. Section 3 reviews related work and positions our contribution. Section 4 formalises the system model and problem definition. Section 5 describes the methodology and instrumentation. Section 6 presents the experimental evaluation. Section 7 reports a validation on real quantum hardware. Section 8 discusses the findings and future directions. Section 9 concludes the paper.

2. Background

This section summarises the technical background required to interpret our system design and measurements.

2.1. Hybrid quantum–classical workloads and variational training

Near-term quantum devices operate in the *Noisy Intermediate-Scale Quantum* (NISQ) regime, where limited qubit counts and noise constrain executable circuit width and depth [1]. A prevalent approach in this regime is the *variational quantum algorithm* (VQA), which couples a parameterised quantum circuit with a classical optimiser [2,18–20]. This hybrid loop underpins much of quantum machine learning (QML), including parameterised quantum models [21,22] and kernel-style feature maps [23]. For learning tasks with classical data, practical feasibility often depends as much on data access patterns and estimator variance as on the expressivity of the quantum model [24].

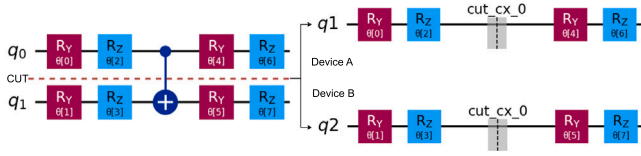


Fig. 1. Independent subcircuits generated by cutting a circuit. The dashed line indicates the cut location; the resulting fragments (left and right subcircuits) can be executed independently on smaller devices. Each cut exponentially increases the number of subexperiments and reconstruction terms.

From a systems perspective, VQA training is dominated by repeated evaluation of expectation values and (often) gradients. Analytic gradient schemes such as the parameter-shift rule are attractive because they avoid finite-difference instability but still multiply circuit evaluations [4]. In addition, the optimisation landscape can be problematic (e.g., barren plateaus), which can increase the number of training iterations and amplify any per-iteration overheads [2,25]. Consequently, end-to-end runtime is governed by a composition of quantum execution latency, the number of circuit evaluations required by the optimiser, and classical orchestration and post-processing costs [1,2].

Noise and sampling error further complicate this picture. Error mitigation can improve estimator accuracy without full fault tolerance, but it typically adds additional circuit executions and classical processing steps [3,5]. The resulting workload is therefore *estimator-heavy*: it comprises large numbers of small quantum jobs plus non-trivial classical aggregation, often executed in an interactive loop [2,3].

2.2. Estimator abstractions and execution models

Near-term applications consume quantum hardware through an expectation-value interface: a parameterised circuit and observable(s) yield an estimate with finite-sample variance. Modern stacks formalise this as first-class execution units [26–29]. These abstractions make performance failure modes predictable: circuit evaluations grow with estimator variance and optimiser query complexity, aggregation is unavoidable, and the critical path may be dominated by orchestration and reduction rather than raw quantum execution [3–5]. We explicitly treat this as a distributed systems problem, measuring and attributing overheads across pipeline stages.

2.3. Circuit cutting and classical reconstruction

Circuit cutting (often discussed under the umbrella of *circuit knitting*) decomposes a circuit that does not fit a target backend into a collection of smaller subcircuits that do, at the cost of classical reconstruction and increased sampling complexity [9–11]. The core trade-off is structural: cutting reduces spatial requirements (qubits) while increasing the number of executions and the weight of post-processing [9, 10] (see Fig. 1).

Practical toolflows typically involve three stages. First, a compiler or optimiser chooses cut locations and a decomposition strategy [10, 12]. Second, a set of subcircuits (often many variants induced by the decomposition) are executed and measured [10,13]. Third, a classical reconstruction procedure aggregates sub-results into an estimate of the target quantity (e.g., an expectation value) [10–12]. Recent work shows that allowing classical communication between subcomponents can reduce overhead constants in certain settings [11], and that alternative cutting schemes based on randomised measurements provide different accuracy–cost trade-offs [13].

Importantly for systems evaluation, cutting turns a single “large” circuit evaluation into a *task graph* with (often) substantial fan-out followed by a global aggregation. This introduces obvious bottleneck candidates: scheduling overhead, straggler sensitivity across large job

sets, and a potentially heavy reduction (reconstruction) stage [10,12]. These are precisely the failure modes we can only resolve by measured profiling rather than by asymptotic overhead counts alone.

The overhead scaling of circuit cutting is exponential in the number of cuts: each additional cut multiplies the number of required subexperiments and reconstruction terms. For c cuts across CNOT gates, the number of subexperiments grows as $\mathcal{O}(9^c)$ in the wire-cutting decomposition employed by *qiskit-addon-cutting* [10,17], and the classical reconstruction cost scales correspondingly. This exponential growth has profound practical consequences that we quantify empirically in our evaluation (Section 6).

How reconstruction works and why it is costly. Because reconstruction is the stage we identify as the dominant bottleneck, we describe its mechanics explicitly. Cutting replaces each non-local operation at a cut by a *quasiprobability decomposition* into locally realisable operations: a two-qubit channel spanning a cut is written as a weighted sum $\sum_a c_a (A_a \otimes B_a)$, where A_a and B_a act only within their respective fragments and the real coefficients c_a may be *negative* [10,11]. For a wire cut realised through CNOT decomposition this expansion contributes a fixed number k of basis terms per cut, and the one-norm of the coefficients, $\gamma = \sum_a |c_a| = 3$, sets the cost of sampling: the resulting sampling overhead is quadratic in γ , that is, $\gamma^2 = 9$ per cut and $\gamma^{2c} = 9^c$ for c independent cuts [11]. After the induced subcircuit variants are executed and the local expectation value of each subobservable is estimated on each fragment f , the target expectation value is reconstructed as a coefficient-weighted sum over the *tensor products* of the per-fragment outcomes,

$$\langle O \rangle \approx \sum_{\mathbf{a}} \left(\prod_{i \in \text{cuts}} c_{a_i} \right) \prod_{f \in \text{fragments}} \langle o_f \rangle_{\mathbf{a}}, \quad (1)$$

where the outer sum runs over all joint assignments $\mathbf{a} = (a_1, \dots, a_c)$ of decomposition terms to the c cut locations. The cost of reconstruction materialises from two sources. First, the number of joint assignments—and hence of coefficient-weighted product terms that must be formed and summed—grows multiplicatively with each cut as $\mathcal{O}(k^c)$, because each cut independently multiplies the number of basis terms to be combined; the volume of measurement data entering this sum grows in step with the $\gamma^{2c} = 9^c$ sampling overhead. Second, for every term the procedure performs a tensor-product combination of fragment results together with bookkeeping over the subobservable and measurement-basis combinations per fragment. Crucially, this is *purely classical* post-processing whose cost is intrinsic to the decomposition and, unlike quantum execution time, does not diminish with improved quantum hardware. This is the structural reason reconstruction can come to dominate the per-query critical path as the number of cuts increases.

2.4. Distributed execution, stragglers, and staged pipelines

Distributed systems research has shown that scalability is frequently capped by coordination and stragglers rather than by idealised parallel work [6,7]. Speculative execution (e.g., LATE [7]), in-memory dataflow [30], and task-graph engines such as Ray [31] address these bottlenecks at different levels. In distributed ML, parameter-server designs [32,33], lock-free updates [34], and collective communication [8, 35] all show that aggregation phases can dominate at scale even when compute parallelises cleanly [36,37]. Circuit cutting produces a staged pipeline with the same fundamental shape—embarrassingly parallel subcircuit execution followed by a global reconstruction barrier—so the relevant question is whether reconstruction and coordination costs dominate the critical path as we scale out [7,8,10]. Our evaluation quantifies this boundary empirically.

3. Related work

We organise related work into (i) circuit cutting methods and their overhead models, (ii) hybrid software stacks and execution interfaces, and (iii) distributed systems techniques for scaling estimator-heavy workloads.

3.1. Circuit cutting methods, overheads, and optimisation targets

Peng et al. [9] introduced a cluster simulation view motivating decomposition as a route to execute circuits beyond device width. CutQC [10] provided an end-to-end toolflow documenting that post-processing and subcircuit counts grow rapidly with cuts. Subsequent work reduced overhead via classical communication [11] and randomised measurements [13], while Harrow et al. [12] formalised optimality notions and bounds. Complementary work on multi-processor execution [38] and serverless cutting environments [39–41] acknowledges that orchestration and classical processing are first-order concerns. However, the literature reports overhead primarily in *counts* rather than *measured* end-to-end time attribution, limiting understanding of when reconstruction becomes the bottleneck and whether scheduling policies materially affect throughput.

3.2. Hybrid stacks and estimator-oriented interfaces

The hybrid execution ecosystem has matured: Qiskit [26] and its primitives [27] make expectation estimation explicit, while PennyLane [28] and TensorFlow Quantum [29] integrate quantum evaluation into differentiable ML workflows. These stacks generally do not analyse distributed bottlenecks for cutting-plus-training pipelines. Error mitigation adds further circuit evaluations and classical processing [3,5], and measurement cost is a central constraint [2,19]. A related line of research is *quantum architecture search* (QAS), which automates the design of the parameterised circuit itself. Recent work uses curriculum reinforcement learning to discover architectures that remain performant under realistic hardware error profiles [42], explicitly targeting the gap between ideal circuit expressivity and device-imposed constraints—the same gap that circuit cutting addresses from the execution side. QAS is orthogonal and complementary to our contribution: it selects *which* circuit to run, whereas we study the distributed systems cost of *executing* a fixed circuit under cutting. A QAS-discovered architecture that exceeds device width would still rely on cutting to run, and would therefore inherit exactly the reconstruction and coordination bottlenecks we characterise; conversely, the per-query overhead profile we measure could serve as a hardware-aware cost term inside a QAS objective. Yet, the interaction between these algorithmic requirements and distributed execution policies is typically left implicit. Our work treats this interaction as the primary object of measurement.

3.3. Distributed ML and straggler-aware scheduling

Distributed ML provides directly relevant methods: parameter-server and bounded-staleness models manage heterogeneity [32,33], speculative execution mitigates tail latency [7], and collective aggregation can dominate at scale [8,35,36]. Classical adversarial robustness work [14, 15,43,44] shows that training dynamics materially affect robustness, and analogous concerns arise in estimator-heavy quantum training from sampling noise and distribution shifts induced by decomposition [2–4]. Existing work rarely connects these robustness questions to the systems bottlenecks of cutting and reconstruction.

3.4. Comparison with existing approaches

Table 1 summarises how our work relates to the most directly comparable circuit-cutting studies across five dimensions relevant to practical deployment.

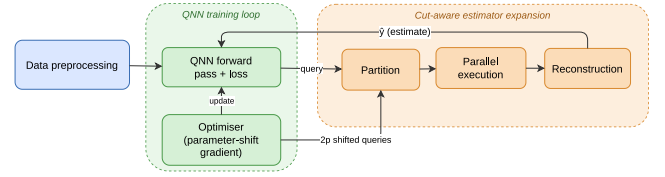


Fig. 2. Training pipeline with a cut-aware distributed estimator, laid out left-to-right. **Left (blue):** classical data preprocessing. **Centre (green):** the QNN training loop, comprising the forward pass/loss and the optimiser. **Right (orange):** the cut-aware estimator expansion—partitioning, parallel execution, and classical reconstruction. Crucially, the expansion is triggered by *both* the forward pass *and* the optimiser: the parameter-shift rule issues $2p$ shifted estimator queries per iteration (for p parameters), each of which independently undergoes the same partition–execute–reconstruct expansion. The reconstructed estimate $\hat{y}$ is returned to the training loop.

3.5. Summary and gap

Prior work establishes that circuit cutting extends executable circuit sizes and clarifies overhead trade-offs [9–13], while hybrid stacks expose estimator-oriented execution [26–29] and distributed systems research provides mature straggler and aggregation techniques [6–8, 32]. What remains underexplored is a measurement-driven account of how cutting transforms end-to-end training into a staged distributed pipeline and which execution policies materially improve throughput without degrading accuracy or robustness. We address this gap by instrumenting costs across cutting, execution, reconstruction, and training dynamics.

4. System model and problem formulation

In this section, we formalise the estimator-driven, cut-aware training pipeline as a staged distributed computation and define measurable performance and outcome quantities aligned with our research questions (RQ1–RQ5).

4.1. End-to-end pipeline model

Fig. 2 shows the pipeline we evaluate. A classical dataset is pre-processed and supplied to a training loop. Each iteration evaluates a quantum neural network (QNN) forward pass and associated loss, and updates parameters via a classical optimiser. The forward pass is implemented via expectation estimation queries of the form $(C(\theta, x), O)$, where C is a parameterised circuit instantiated at parameters θ and input x , and O is the observable (or set of observables) required to compute model outputs and loss [2,4].

When circuit cutting is enabled, the estimator query is expanded into a set of subexperiments that can be executed independently, followed by a classical reconstruction step that returns the estimate to the training loop [9,10,12]. This transformation is central to our study because it introduces (i) additional overheads beyond baseline execution (RQ1), (ii) a reconstruction barrier that can dominate the critical path under scale-out (RQ2), and (iii) scheduling degrees of freedom, including staggering, that affect straggler sensitivity and throughput (RQ3). The same transformation may also affect learning dynamics through estimator variance and evaluation order, which we examine through accuracy and robustness outcomes (RQ4–RQ5).

4.2. Integration with QML training loop

To make the integration between the cut-aware pipeline and the QML training loop precise, we describe the data and control flow through the key software components.

Table 1

Comparison with key related work on circuit cutting. **Approach:** theoretical analysis (T), end-to-end toolflow (TF), or systems profiling (SP). **Overhead analysis:** asymptotic bounds (A), empirical measurement (E), or both. **Training integration:** whether cutting is embedded in an iterative variational training loop. **Hardware evaluation:** whether results on a real QPU are reported. **Bottleneck identification:** whether specific pipeline stages are measured and identified as scaling limiters.

Work	Approach	Overhead analysis	Training integration	Hardware evaluation	Bottleneck identification
Peng et al. [9]	T	A	×	×	×
Tang et al. (CutQC) [10]	TF	A+E	×	✓	×
Piveteau & Sutter [11]	T	A	×	×	×
Lowe et al. [13]	T	A	×	×	×
Harrow & Lowe [12]	T	A	×	×	×
Carrera V. et al. [38]	TF	E	×	✓	×
Our work	SP	E	✓	✓	✓

The parameterised quantum circuit $C(\theta, x)$ and observable O are encapsulated in an *EstimatorQNN* instance [16], which exposes a differentiable forward pass and gradient computation interface. The *TorchConnector* wraps this *EstimatorQNN* as a PyTorch `nn.Module`, enabling standard PyTorch optimisers (e.g., Adam) and loss functions (e.g., cross-entropy) to be used without modification.

During the forward pass, *TorchConnector* calls *EstimatorQNN.forward()*, issuing estimator queries $(C(\theta, x_i), O)$ for each sample x_i . In the baseline configuration, queries are evaluated directly by the Aer simulator; when cutting is enabled, *DistributedEstimator* intercepts each query and processes it through the staged pipeline below.

For gradient computation, the *EstimatorQNN* uses the parameter-shift rule [4]: two additional queries with $\theta_j \pm \pi/2$ per parameter θ_j , each independently processed through the cut-aware pipeline. The cut-aware estimator is transparent to the optimiser: gradients are mathematically equivalent to the uncut case up to finite-sample variance. The overhead is multiplicative: the total cut-aware estimator invocations per iteration is $(q + 2p) \times [\text{batch}]$ (for q forward queries and p parameters), each expanding into K subexperiments—the root cause of the training time overheads documented in RQ1.

4.3. Cut-aware estimator query model

For a single query instance $(C(\theta, x), O)$, circuit cutting produces a partition labelled by ℓ and yields subcircuits and subobservables. The resulting execution requires a finite set of concrete subexperiments $\mathcal{E} = \{e_k\}_{k=1}^K$ and reconstruction coefficients α determined by the cutting scheme and partition [10,12]. Each e_k is executed with a fixed shot count S , producing a classical summary r_k (e.g., outcome counts). Reconstruction aggregates $\mathcal{R} = \{r_k\}$ into an estimate $\hat{y}$ of the target expectation value.

This induces a staged cost decomposition for each query instance:

$$T_{\text{total}} = T_{\text{part}} + T_{\text{gen}} + T_{\text{exec}} + T_{\text{rec}}, \quad (2)$$

where:

- T_{part} is circuit/observable partitioning time (including cut planning and partition materialisation);
- T_{gen} is subexperiment generation time (including basis variants and reconstruction coefficients);
- T_{exec} is parallel execution time of $\mathcal{E}$ on a worker pool;
- T_{rec} is classical reconstruction time.

Eq. (2) provides the measurement backbone for RQ1 and RQ2: we report each component as a measured quantity and study how their shares evolve with the degree of parallelism.

Amortisation of structural stages across training. Eq. (2) decomposes the cost of a *single* query, but in an iterative training loop the circuit *structure* $C(\cdot, \cdot)$ —its gates, wires, and cut locations—is invariant across queries; only the parameter values (θ, x) change between iterations. Consequently, partitioning and the generation of subexperiment

templates (together with the reconstruction coefficients α , which depend only on the cut structure) need to be computed *once* for a given partition label and then reused. Our implementation caches these structural artefacts keyed on the circuit structure and partition label, so that for every subsequent query partitioning and generation are not repeated: each query instead pays only a lightweight parameter-binding cost T_{bind} that rebinds (θ, x) into the cached templates. The per-query cost over a training run of N queries is therefore more precisely

$$T_{\text{total}}^{(j)} = \underbrace{(T_{\text{part}} + T_{\text{gen}}) \mathbf{1}\{j = 1\}}_{\text{one-time, cached}} + T_{\text{bind}}^{(j)} + T_{\text{exec}}^{(j)} + T_{\text{rec}}^{(j)}, \quad (3)$$

for the j th query ($j = 1, \dots, N$), so that the amortised contribution of $T_{\text{part}} + T_{\text{gen}}$ vanishes as $1/N$ over a training run. This corrects a redundancy that would otherwise artificially inflate the per-iteration overhead by re-deriving structural quantities at every step. For the small circuits studied here, T_{part} and T_{gen} are already small relative to $T_{\text{exec}} + T_{\text{rec}}$, so caching leaves the reported stage *shares* essentially unchanged; for larger and deeper circuits, where cut planning and template generation become expensive (Section 6), this amortisation is essential and shifts the relevant per-iteration cost onto T_{bind} , T_{exec} , and T_{rec} .

4.4. Execution, stragglers, and scheduling

We model subexperiment execution as a set of independent tasks scheduled on w workers, followed by a reconstruction barrier. Fig. 3 captures this per-query expansion and the straggler sensitivity introduced by the barrier. Let t_k denote the service time for subexperiment e_k (including dispatch and backend runtime). Under a work-conserving policy, the realised parallel time is governed by load balance and tail effects and can be expressed as:

$$T_{\text{exec}} \approx \max_{i \in \{1, \dots, w\}} \sum_{k \in \mathcal{A}(i)} t_k, \quad (4)$$

where $\mathcal{A}(i)$ is the assignment of subexperiments to worker i . This form makes explicit why speed-up can saturate even when the task set is abundant: T_{exec} is bounded by the slowest worker, and T_{rec} is a serial reduction that must wait for all results, mirroring classical bottlenecks in staged distributed computations [6,7].

To study straggler sensitivity repeatedly, we allow an optional synthetic delay process at the task level. Each subexperiment independently incurs an injected delay Δ with probability p , yielding $t'_k = t_k + \mathbf{1}\{u_k < p\} \Delta$ for $u_k \sim \text{Uniform}(0, 1)$. We use this mechanism strictly as an experimental control to stress the tail of Eq. (4) and to evaluate whether scheduling choices (including staggering) reduce end-to-end time-to-solution under heterogeneity [7].

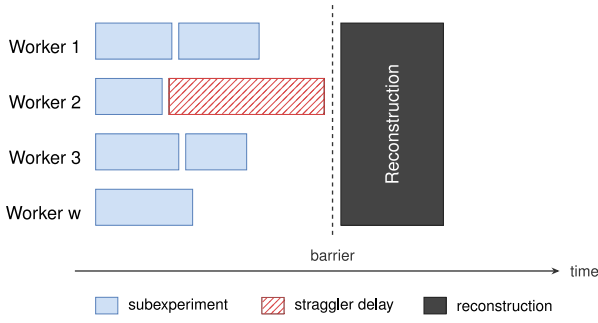


Fig. 3. Per-query expansion under circuit cutting. Subexperiments (light shaded blocks) execute across w workers; classical reconstruction (dark block) forms a barrier that cannot begin until all subexperiment results are available. A straggler delay (hatched block, Worker 2) extends that worker's completion time and therefore pushes back the barrier, delaying reconstruction and lengthening the critical path.

4.5. Problem definition

We consider a fixed training protocol (dataset split, optimiser, stopping criterion) and a fixed per-subexperiment shot budget S . For a given configuration (cut plan ℓ , worker pool size w , scheduling policy π , and optional straggler parameters), our primary systems objective is to minimise *time-to-solution* while preserving learning outcomes:

$$\min_{\pi \in \Pi} \mathbb{E}[T_{\text{total}}] \quad \text{s.t.} \quad \text{accuracy and robustness constraints.} \quad (5)$$

We operationalise this objective through:

- **Measured:** $T_{\text{part}}, T_{\text{gen}}, T_{\text{exec}}, T_{\text{rec}}, T_{\text{total}}$ per query instance (and their aggregation over training);
- **Derived:** stage shares (e.g., $T_{\text{rec}}/T_{\text{total}}$), scaling curves as a function of w , and bottleneck attribution via changes in dominant stage across configurations;
- **Observed:** training outcomes (accuracy) and robustness metrics computed under a consistent evaluation regime.

This mapping supports RQ1–RQ5: overhead and scaling from timing breakdowns, reconstruction bottleneck from measured T_{rec} dominance and speed-up saturation, staggering impact from T_{exec} tail behaviour changes, and outcome preservation from accuracy and robustness under unchanged evaluation procedures.

5. Methodology

In this section, we describe how we implement the cut-aware distributed estimator, how we instrument stage-level timings and outcomes, and how we configure scheduling policies and straggler controls to answer RQ1–RQ5.

5.1. Cut-aware distributed estimator

Our estimator backend implements a cut-aware execution path that transforms each query instance $(C(\theta, x), O)$ into a set of subexperiments and a reconstruction plan. Concretely, for each query instance we: (i) partition the problem according to a partition label ℓ , (ii) generate the concrete subexperiments $\mathcal{E}$ and reconstruction coefficients α , (iii) execute $\mathcal{E}$ across a worker pool of size w , (iv) reconstruct the target estimate $\hat{y}$, (v) return $\hat{y}$ to the training loop.

The key methodological choice is to treat this as a *staged* pipeline and to instrument each stage individually. This allows us to attribute overheads introduced by cutting (RQ1) and to identify scaling limits caused by reconstruction and coordination (RQ2) without conflating them with training-loop compute.

Algorithm 1 Instrumented cut-aware estimator for one query instance

Require: Circuit $C(\theta, x)$, observable O , partition label ℓ , QPD sample budget M , measurement shots S , workers w , scheduling policy π

- 1: Initialise timers and metadata record
- 2: **if** cut plan for $(C\text{-structure}, \ell)$ not cached **then**
- 3: **Partition:** $(\{C_i\}, \{O_i\}, B) \leftarrow \text{PartitionProblem}(C, O, \ell)$
- 4: **Generate:** $(\mathcal{E}, \alpha) \leftarrow \text{GenerateSubexperiments}(\{C_i\}, \{O_i\}, M)$
- 5: Cache $(\mathcal{E}, \alpha, \{O_i\})$ keyed on $(C\text{-structure}, \ell)$
- 6: **end if**
- 7: **Bind:** $\mathcal{E}_{\theta, x} \leftarrow \text{BindParameters}(\mathcal{E}, \theta, x)$
- 8: **Execute:** $\mathcal{R} \leftarrow \text{ExecuteTasks}(\mathcal{E}_{\theta, x}, S, w, \pi)$
- 9: **Reconstruct:** $\hat{y} \leftarrow \text{Reconstruct}(\mathcal{R}, \alpha, \{O_i\}, O)$
- 10: Log $(T_{\text{part}}, T_{\text{gen}}, T_{\text{bind}}, T_{\text{exec}}, T_{\text{rec}}, T_{\text{total}})$ and metadata to JSONL
- 11: **return** $\hat{y}$

Algorithm 1 summarises the per-query logic. The algorithm is intentionally phrased to align with logged quantities and the timing decomposition in Eq. (2).

Three aspects of Algorithm 1 warrant clarification. First, $B = \{\beta_i\}$ is the set of quasiprobability *bases* produced by partitioning: each cut contributes one basis β_i with an individual sampling overhead γ_i , and the total sampling overhead of the cut plan is the product $\prod_i \gamma_i$ (e.g. 9^c for c CNOT-based wire cuts). B thus determines the per-query overhead we report and is otherwise consumed by generation. Second, the generation step takes the QPD sample budget M —the number of quasiprobability samples used to represent the decomposition of the cut channels—which is distinct from the *measurement shots* S applied when each subexperiment is executed on the backend (step 8). The two are commonly conflated: M controls how faithfully the quasiprobability decomposition is represented and may be taken to be exact ($M \rightarrow \infty$), whereas S controls per-circuit sampling variance. Third, because partitioning and generation depend only on the circuit *structure* and partition label—not on (θ, x) —they are computed once and cached (steps 2–6), and every subsequent query reuses the cached templates and pays only the parameter-binding cost of step 7, consistent with the amortised cost model in Eq. (3).

5.2. Scheduling policies and staggering

We evaluate execution policies that differ in how they submit the task set $\mathcal{E}$ to a bounded worker pool of size w . We consider three dispatch strategies: (1) **Eager dispatch**, where all subexperiments are submitted immediately under work-conserving, first-come-first-served allocation (our baseline); (2) **Batched dispatch**, where subexperiments are partitioned into batches of size B with inter-batch delay δ to control concurrency; and (3) **Staggered dispatch**, where subexperiments are grouped by structural similarity and spaced to avoid correlated queueing delays. All three terminate in the same reconstruction barrier. A policy that reduces task-completion variance can lower $\max_i \sum_{k \in \mathcal{A}(i)} t_k$ and thereby reduce T_{exec} without changing total work.

Our current evaluation focuses on eager dispatch with straggler injection to stress the tail of T_{exec} (RQ3). A systematic comparison of batched and staggered policies is an important future direction; the instrumentation framework supports this extension, with all dispatch parameters logged alongside stage-level timings.

5.3. Straggler control

When enabled, synthetic straggler injection is applied at task execution time. Each subexperiment independently incurs an additional delay Δ with probability p , yielding $t'_k = t_k + \mathbf{1}\{u_k < p\}\Delta$. This allows us to stress the barrier sensitivity in Fig. 3 and evaluate whether the critical-path structure (execution-dominated vs. reconstruction-dominated) modulates straggler impact (RQ3). We log injected delays alongside per-stage timings to ensure that straggler effects are explicitly traceable in subsequent analysis.

5.4. Instrumentation and outcome tracking

For each estimator query, we emit a JSONL record containing query structure (K, S), configuration (ℓ, w , scheduling policy, straggler parameters), and stage timings per Eq. (2). Training-level logs capture loss traces, optimiser steps, and evaluation metrics. This supports overhead attribution (RQ1), scaling saturation analysis (RQ2), straggler sensitivity quantification (RQ3), and outcome preservation assessment (RQ4–RQ5) under controlled, matched comparisons.

6. Performance evaluation

We evaluate the proposed cut-aware estimator pipeline using instrumented execution traces to quantify end-to-end overheads, scaling behaviour, sensitivity to injected stragglers, and learning outcomes (accuracy and robustness). All claims in this section are grounded in measured quantities recorded in the run-level records, estimator-call summaries, and robustness traces.

6.1. Experimental setup

Software stack. All experiments are implemented in Python. Quantum circuits are constructed in Qiskit [45] and executed using the Aer primitive interface (SamplerV2/AerSampler, depending on the installed *qiskit-aer* version). Circuit cutting is implemented with *qiskit-addon-cutting* [17] via the standard decomposition workflow, which performs wire cutting through CNOT gate decomposition. The learning pipeline uses *qiskit-machine-learning* (EstimatorQNN and TorchConnector [16]) and PyTorch for optimisation; MNIST [46] data handling uses Torchvision, and Iris [47] preprocessing uses scikit-learn. The distributed execution model uses a bounded thread-based worker pool, with the worker count logged as *subexp_workers* (MNIST) or *max_subexp_workers* (Iris).

Hardware platform. The classical host for all simulator-based experiments and for the classical reconstruction stage is an Apple M4 system-on-chip (10 cores) with 16 GB of unified memory running macOS 26.5. The full software stack comprises Python 3.10, Qiskit 2.3.1, *qiskit-aer* 0.17.2, *qiskit-machine-learning* 0.9.0, *qiskit-ibm-runtime* 0.46.1, and PyTorch 2.11. For the real-hardware validation reported in Section 7, subexperiments are executed on the IBM Quantum *ibm_kingston* superconducting processor (an IBM Heron r2 device, 156 qubits) accessed through the *qiskit-ibm-runtime* Sampler primitive; circuits are transpiled to the device basis with a preset pass manager at optimisation level 1, and reconstruction is performed on the classical host above. We report the device backend, transpilation settings, and shot budget alongside every hardware measurement so that platform-dependent timing effects are explicitly attributable.

Model circuits. Across both workloads, we instantiate a parameterised quantum circuit as the composition of a *ZFeatureMap* (encoding classical features into qubit rotations) followed by a *RealAmplitudes* ansatz with one repetition (*reps*=1), which provides a hardware-efficient parameterisation of the unitary search space. For Iris, the observable is the tensor-product Pauli-Z operator $Z^{\otimes n}$, implemented as a *SparsePauliOp* with n qubits, yielding a scalar expectation-value classifier output. For MNIST, a similar observable structure is used. The choice of a single-repetition ansatz keeps the parameter count manageable while providing sufficient expressivity for the binary classification tasks considered.

Workloads and training budgets. We consider two binary classification tasks: *iris_binary.pm1* (two classes from the Iris dataset, labels mapped to ± 1) and *mnist_binary* (two-digit binary classification from MNIST). We enforce matched comparisons within each dataset by fixing the estimator shot budget (1024 shots per (sub)experiment) and by restricting comparisons to runs executed under the same training budget and straggler configuration.

Table 2

Planning-stage microbenchmark (median over repeated trials, no execution). Partitioning T_{part} scales mildly with circuit width and is independent of cut count; template generation T_{gen} tracks the $\mathcal{O}(9^c)$ subexperiment count K . Both are one-time, cacheable structural costs (Eq. (3)).

Qubits	Parts	Cuts	K	T_{part} (ms)	T_{gen} (ms)
4	2	1	12	1.3	2.7
8	2	1	12	1.5	4.1
12	2	1	12	2.1	5.7
16	2	1	12	2.4	7.3
12	3	2	108	2.1	33.1
12	4	3	864	2.4	218.7

For the clean (non-straggler) setting, we report results using the longest training budgets available for each dataset (Iris: *maxiter* = 60; MNIST: 10 epochs), with *straggler_delay_s* = 0.0. For scaling analysis, shorter budgets are used (Iris: *maxiter* = 10; MNIST: 5 epochs), again with *straggler_delay_s* = 0.0, to match the worker-count sweeps used in our scaling experiments. For straggler sensitivity, we inject delays of *straggler_delay_s* = 0.1; these experiments use Iris with *maxiter* = 10 and MNIST with 3 epochs at 8 workers.

Computational cost and qubit-count scope. The computational cost of training hybrid quantum models with circuit cutting grows exponentially with the number of cuts. In the wire-cutting decomposition used by *qiskit-addon-cutting*, each cut across a CNOT gate decomposes the gate into nine subexperiment variants, yielding a multiplicative overhead of $\mathcal{O}(9^c)$ for c cuts [10,17]. This means that a circuit with one cut requires $9\times$ more subexperiments than the uncut baseline, two cuts require $81\times$, and three cuts require $729\times$. In addition to the subexperiment count, the classical reconstruction cost and the intermediate state that must be stored and processed grow correspondingly.

This constraint is a property of *cut-aware training*, not of circuit cutting in isolation. Single-shot circuit-cutting toolflows such as CutQC routinely target much larger circuits—on the order of tens to over a hundred qubits—because they evaluate a circuit *once* [10,12]. Our setting differs fundamentally: we embed cutting inside an *iterative* variational training loop. As established in Section 4.2, each training run issues $(q+2)p \times |\text{batch}|$ estimator queries *per iteration* (the parameter-shift gradient contributes $2p$ queries for p parameters), and *every* such query independently expands by the $\mathcal{O}(9^c)$ factor. The quantity that becomes prohibitive is therefore the *product* of the cutting overhead and the training-query volume, not the cutting overhead alone. It is this product—compounded across iterations, dataset samples, seeds, and worker-count sweeps—that restricts systematic experimentation to small circuit widths, even though a single cut evaluation at these cut counts is itself inexpensive.

To separate the *structural* (one-time, cacheable) cost from the per-query execution and reconstruction work, we microbenchmark the planning stages T_{part} (partitioning) and T_{gen} (subexperiment-template generation, exact quasiprobability representation $M \rightarrow \infty$) in isolation, with no circuit execution. Table 2 reports median times for the *ZFeatureMap*+*RealAmplitudes* circuit used throughout. Partitioning is cheap and grows only mildly with width (1.3 ms at 4 qubits to 2.4 ms at 16 qubits, single cut), so *cut planning* is not the bottleneck and scales well beyond the widths we train on. Template generation tracks the $\mathcal{O}(9^c)$ subexperiment count (5.5 ms, 33 ms, 219 ms at one, two, three cuts for 12 qubits). Because these artefacts depend only on circuit structure and partition label, they are computed once and cached (Eq. (3)), amortised over the thousands of parameter-binding queries in a run rather than recurring per iteration. The genuine per-query bottleneck is therefore the execution and reconstruction stages, which cannot be amortised because they depend on the live parameters (θ, x) —motivating the restriction to small widths where these stages can be swept systematically.

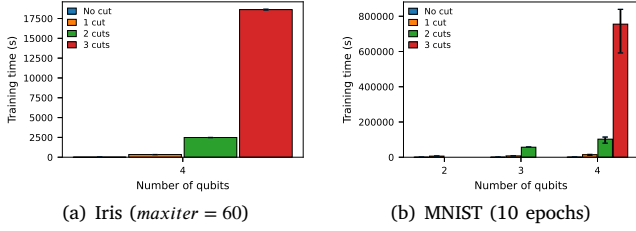


Fig. 4. RQ1: End-to-end training time under clean execution. Training time increases substantially with cut count due to subexperiment expansion and reconstruction overhead.

This exponential scaling renders systematic *training* experiments beyond 4 qubits computationally prohibitive. We therefore restrict our evaluation to circuits with up to 4 qubits. This is a scope boundary imposed by the iterative training loop and the requirement for controlled, repeated sweeps—not a claim that circuit cutting itself is limited to this scale. Our findings about reconstruction dominance and scaling saturation are structural properties that become more pronounced at larger scales.

Logging and derived metrics. Each run records end-to-end training time, test accuracy, stage-level timings per estimator call, and robustness traces under Gaussian noise and FGSM perturbations. We report reconstruction fractions, scaling ratios, and straggler sensitivity computed only between matched runs (dataset, seed, configuration, and cut label) to ensure observed differences are attributable to the varied parameter.

6.2. Results and analysis

6.2.1. RQ1: Cutting overhead

Fig. 4 reports end-to-end training time under the clean setting (Iris: $\text{maxiter} = 60$; MNIST: 10 epochs). Training time increases sharply with cut count, consistent with the $\mathcal{O}(9^c)$ subexperiment expansion compounded across all $(q + 2p) \times |\text{batch}|$ estimator queries per iteration. For MNIST, the increase from no cut to three cuts is approximately an order of magnitude (729× more subexperiments per query plus corresponding reconstruction growth). On Iris, the relative overhead is substantial but lower in absolute terms.

6.2.2. RQ2: Scalability bottlenecks

Fig. 5 reports speed-up from increasing parallelism (ratio of training time at 1 worker to 16 workers, matched seed-configuration pairs). Speed-ups are close to or below 1 for both datasets, indicating that additional workers yield negligible time reductions.

The worker pool size w controls only concurrent subexperiment execution; it does not affect serial reconstruction. When T_{rec} dominates, increasing w has diminishing returns—a direct instance of Amdahl’s law. Table 3 quantifies this: reconstruction accounts for 43% (1 cut), 49% (2 cuts), and 53% (3 cuts, 95th percentile 58%) of per-query time. Even with perfect execution parallelisation, total query time at three cuts could be reduced by at most 47%, explaining the scaling saturation in Fig. 5.

6.2.3. RQ3: Straggler sensitivity

Fig. 6 reports sensitivity to injected stragglers ($\text{straggler_delay}_s = 0.1$), showing the slowdown ratio at $p = 0.2$ relative to $p = 0.0$. Reconstruction-dominated configurations exhibit *lower* relative sensitivity to execution-side stragglers, since T_{rec} is unaffected by execution delays and the additional straggler time is a smaller fraction of the already-large T_{total} . Conversely, execution-dominated configurations amplify tail latency, because the reconstruction barrier waits for all subexperiments and the slowest straggler determines T_{exec} .

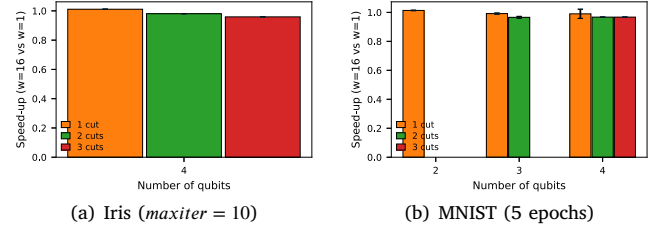


Fig. 5. RQ2: Scaling behaviour under clean execution. Bars report speed-up at 16 workers relative to 1 worker for matched pairs. Speed-up near or below 1 indicates reconstruction overhead dominates.

Table 3

RQ2: Reconstruction share ($T_{\text{rec}}/T_{\text{total}}$) from estimator-call summaries under clean execution ($\text{straggler_delay}_s = 0.0$). The monotonic increase with cut count confirms that reconstruction progressively dominates per-query time.

#cuts	n	median $T_{\text{rec}}/T_{\text{total}}$	p95 $T_{\text{rec}}/T_{\text{total}}$
1	41	0.430	0.474
2	28	0.489	0.533
3	5	0.530	0.580

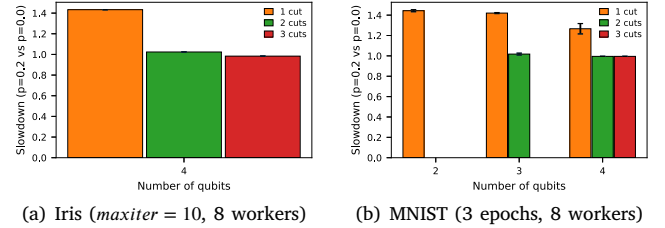


Fig. 6. RQ3: Straggler sensitivity at $\text{straggler_delay}_s = 0.1$. Bars report slowdown at $p = 0.2$ relative to $p = 0.0$. Reconstruction-dominated configurations show lower sensitivity to execution-side stragglers.

Straggler mitigation (speculative execution [7], work stealing [37]) is therefore most effective when T_{exec} dominates; in reconstruction-dominated regimes, effort is better directed at reducing or overlapping reconstruction.

6.2.4. RQ4: Accuracy

Fig. 7 reports test accuracy under the clean setting with the maximum training budgets (Iris: $\text{maxiter} = 60$; MNIST: 10 epochs).

On Iris, all configurations achieve identical test accuracy, confirming that cutting does not degrade predictive performance: reconstruction is mathematically exact up to finite-sample variance, and the low-dimensional landscape is insensitive to the additional estimator noise.

On MNIST, accuracy varies with qubit descriptor and random seed but shows no systematic degradation with cut count: several cut configurations match or exceed the uncut baseline. Cutting changes the effective noise profile of the estimator, which can alter optimisation trajectories and occasionally improve final accuracy through implicit regularisation, even though the expected value of the reconstructed estimate remains unbiased.

6.2.5. RQ5: Robustness

We evaluate robustness using perturbation traces under two complementary attack models: additive Gaussian noise (random perturbations to input features) and Fast Gradient Sign Method (FGSM) attacks [44] (adversarial perturbations along the gradient direction). Fig. 8 reports

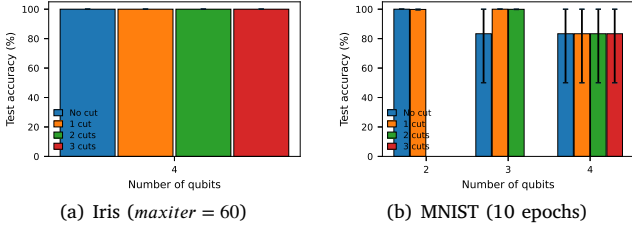


Fig. 7. RQ4: Absolute test accuracy under clean execution. On Iris, accuracy is invariant across cut settings. On MNIST, accuracy is not systematically degraded by cutting.

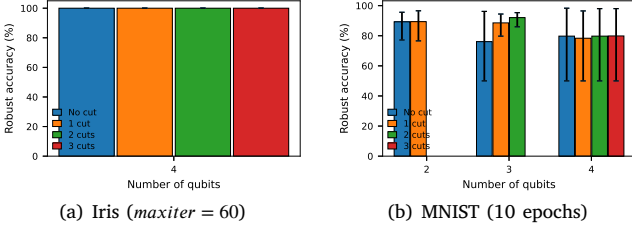


Fig. 8. RQ5: Robustness summary under clean execution. Bars report mean accuracy over non-zero perturbation magnitudes (Gaussian and FGSM). Robustness is not systematically degraded by cutting.

a robustness summary per run, computed as the mean accuracy over non-zero perturbation magnitudes averaged across both attack types.

For Iris, the robustness summary matches clean accuracy across all configurations, consistent with the accuracy invariance in RQ4. For MNIST, robustness varies across configurations, but several cut settings match or exceed the baseline, consistent with the noise-regularisation interpretation: increased estimator variance during training may produce smoother decision boundaries [14]. Given the limited seed count, we report these as measured outcomes; a more extensive evaluation would be needed to establish robustness improvement as a reliable effect.

7. Validation on real quantum hardware

Our main study uses a high-throughput simulator backend so that we can isolate the cut-aware pipeline structure (overheads, scaling, straggler sensitivity) from the confounding latencies of a shared hardware queue, and so that we can afford the large number of repeated estimator queries an iterative training loop demands. This design choice has an important consequence: on a local simulator the per-circuit execution time is very small, so the measured execution stage T_{exec} is compressed relative to what a real device would incur, and the relative share of reconstruction is correspondingly inflated. To test whether our findings transfer to physical hardware—and to characterise how the stage balance changes in a latency-bound regime—we performed a targeted validation on a real superconducting QPU.

Validation protocol. Because real-hardware quantum time is a scarce, metered resource, executing the full iterative training loop on a QPU (every forward and parameter-shift query, for every sample and iteration, each expanded by 9°) is infeasible within a realistic allocation. We therefore adopt a *train-on-simulator, evaluate-on-hardware* protocol: model weights are trained to convergence on the statevector simulator (which, as established in Sections 4–6, produces gradients mathematically equivalent to the uncut model up to finite-sample variance), and the trained model is then evaluated on the QPU under each cut configuration. This isolates the quantity of interest—the *execution* of cut-induced subexperiments and their classical reconstruction on real

hardware—while keeping the hardware budget tractable. We evaluate the Iris binary task with a 4-qubit ZFeatureMap+RealAmplitudes circuit on `ibm_kingston` (IBM Heron r2, 156 qubits), with 1024 shots per subexperiment, circuits transpiled at optimisation level 1, and reconstruction performed on the classical host of Section 6.1. We report clean test accuracy and robustness (Gaussian and FGSM at magnitude 0.1) on both the simulator and the QPU, alongside the hardware stage timings.

Findings. Table 4 summarises the results. Three observations stand out. (i) *Functional correctness on hardware.* Reconstruction from physically measured subexperiment outcomes reproduces the correct predictions: QPU clean accuracy matches the simulator (100% for every configuration on this task), and robustness under both Gaussian and FGSM perturbations is likewise preserved (100%). Circuit cutting therefore composes correctly with real-device noise at this scale—the decomposition and quasiprobability reconstruction are not silently corrupted by hardware error. (ii) *The stage balance inverts on hardware.* Whereas on the simulator execution is near-instantaneous, on the QPU the execution stage becomes the heavy stage—dominated by per-circuit device time, transpilation, and queueing latency—and grows sharply with the subexperiment count: the number of executed circuits scales with the $\mathcal{O}(9^\circ)$ overhead (from the uncut baseline to one cut and beyond), and the measured execution wall-time and on-device run-time rise accordingly, while the classical reconstruction time remains small (tens to a few hundred milliseconds). Subexperiments here were submitted sequentially in chunks to a single backend, so the reported T_{exec} reflects single-device execution plus queueing; dispatching the independent subexperiments across parallel QPUs would reduce this term—a direct hardware realisation of the worker-pool model that we leave to future work. (iii) *The two regimes are complementary, and our architectural conclusion is unchanged.* The 9° growth inflates both stages—more subexperiments to execute and more coefficient-weighted product terms to reconstruct. The simulator isolates the reconstruction growth (a purely classical cost that better quantum hardware cannot reduce); the hardware exposes the execution-latency growth (which better hardware and queueing can reduce). The central message therefore holds under either backend: cutting transforms training into a staged distributed workload whose critical path is shaped by an exponentially growing subexperiment set, and whichever stage dominates, the bottleneck is a systems-level property of the expansion rather than an artefact of the backend.

Incorporating realistic execution delays. The methodology accommodates real-world execution cost in two ways without requiring full-loop hardware runs. First, the synthetic straggler-injection mechanism of Section 5 already perturbs T_{exec} at the task level; the per-circuit run-times measured here (Table 4) and historical device-execution statistics published by QPU providers can parameterise the injected-delay distribution, yielding a hardware-calibrated execution model on top of the simulator pipeline. Second, the amortisation analysis of Eq. (3) continues to hold on hardware: the structural stages are computed once and cached, so the recurring per-iteration cost is the parameter-binding, execution, and reconstruction terms, exactly the quantities the hardware validation measures.

Scope. This validation is deliberately small: it is an inference-time evaluation under fixed weights on a limited test set and a single backend, and it does not perform hardware-based gradient training. Accordingly, the accuracy- and robustness-preservation conclusions of RQ4–RQ5 remain established on the simulator; the hardware experiment confirms functional correctness under real device noise and characterises the execution/reconstruction balance in a latency-bound regime, rather than providing a full hardware scaling study, which we identify as future work (Section 8).

Table 4

Real-hardware validation on `ibm_kingston` (IBM Heron r2, 156 qubits), Iris binary task, 4 qubits, 1024 shots, train-on-simulator/evaluate-on-hardware. K is the number of executed subexperiment circuits per evaluation round; $T_{\text{exec}}^{\text{wall}}$ includes queueing and chunked submission, t_{run} is on-device execution time, T_{rec} is classical reconstruction. Accuracy and robustness are identical on simulator and QPU.

Config	K	$T_{\text{exec}}^{\text{wall}}$ (s)	t_{run} (s)	T_{rec} (s)	QPU Acc.
No cut (AAAA)	10	10.73	0.00	0.028	100%
1 cut (AABB)	120	52.52	9.27	0.195	100%
2 cuts (AABC)	1080	458.84	367.45	1.676	100%

8. Discussion and future work

Our evaluation demonstrates that circuit cutting should be treated as a staged distributed workload rather than a purely algorithmic transformation. The central observation across RQ1–RQ3 is that, while cutting increases parallelisable work, it simultaneously introduces substantial non-parallel components—particularly reconstruction—that dominate end-to-end performance. This section consolidates these findings, discusses the practical implications, and outlines future directions.

8.1. Discussion

Reconstruction is the primary scaling limiter. RQ2 quantifies reconstruction as a substantial and growing share of per-query time (53% median at three cuts), inducing two scaling constraints. First, the serial reconstruction stage bounds achievable speed-up even under ideal execution parallelism—a direct instance of Amdahl’s law applied to the cut-aware pipeline. Amdahl’s law [48] states that if a fraction f of a workload is inherently serial and the remaining $(1 - f)$ is perfectly parallelisable across w workers, the overall speed-up is bounded by $1/(f + (1 - f)/w)$, which converges to $1/f$ as $w \rightarrow \infty$; here the serial fraction f is the reconstruction barrier, which cannot be overlapped with execution in the unoptimised pipeline. At $f = 0.53$ (three cuts, median), this yields a maximum theoretical speed-up of approximately 1.89 $\times$, regardless of how many workers are available. Second, reconstruction acts as a global barrier: the training loop does not progress until all subexperiment results are available, making the pipeline sensitive to upstream tail latency. These effects explain the speed-up saturation observed in Fig. 5 and indicate that reducing the reconstruction fraction is a prerequisite for meaningful parallel scaling.

Reconstruction can be partially overlapped and parallelised. Our current implementation treats reconstruction as a single serial reduction that begins only after a global barrier on all subexperiment results, which is the behaviour measured throughout RQ2. However, the structure of Eq. (1) shows that this barrier is not fundamental. Because the target estimate is a sum over coefficient-weighted tensor products of *per-fragment* outcomes, two forms of relaxation are feasible. (i) *Overlap with execution*: partial products can be accumulated incrementally as fragment results arrive, so reconstruction of the terms whose fragments have already completed can proceed while other subexperiments are still executing, hiding part of T_{rec} behind T_{exec} rather than serialising after it. (ii) *Intra-reconstruction parallelism*: the $\mathcal{O}(9^c)$ product terms in Eq. (1) are mutually independent and can be evaluated concurrently and combined by a parallel (tree) reduction, lowering the span of the summation from linear to logarithmic in the number of terms. Neither relaxation reduces the exponential *work* of reconstruction—that is intrinsic to the decomposition—but both reduce its contribution to the critical path and therefore relax the Amdahl bound above by shrinking the effective serial fraction f . We did not implement these optimisations in the present study because our goal was to *measure* the unoptimised pipeline and establish reconstruction as the bottleneck; realising overlapped and tree-reduced reconstruction is a concrete and, we expect, high-value engineering direction, which we identify as future work below.

The $\mathcal{O}(9^c)$ overhead compounds across training. The exponential overhead is incurred once per estimator query, not once per run. Including the $2p$ parameter-shift gradient queries, the total subexperiments per epoch scale as $\mathcal{O}(9^c \cdot p \cdot |\text{dataset}|)$ —the primary reason training time increases by an order of magnitude at three cuts.

Straggler sensitivity is modulated by critical-path structure. RQ3 shows that reconstruction-dominated configurations exhibit lower sensitivity to execution-side delays, while execution-dominated configurations amplify tail latency. The optimal mitigation strategy therefore depends on where the bottleneck lies: classical straggler mitigation in execution-dominated regimes, reconstruction optimisation in reconstruction-dominated regimes.

Learning outcomes are preserved under cutting. RQ4 confirms that accuracy is fully preserved on Iris and not systematically degraded on MNIST. RQ5 demonstrates that robustness under Gaussian noise and FGSM perturbations is maintained, with some cut settings exhibiting comparable or improved robustness. These findings validate that circuit cutting can serve as a systems-level execution strategy without silently degrading model quality or creating exploitable vulnerabilities in decision boundaries.

Implications for system design. Our findings suggest that: (i) reconstruction should be treated as a first-class optimisation target; (ii) scheduling policies should adapt to the reconstruction fraction; (iii) resource allocation should balance execution workers and reconstruction compute; and (iv) the exponential overhead profile should inform decisions about when cutting is preferable to alternatives (model compression, shallower circuits, or waiting for larger devices).

8.2. Scope and generalisability

We identify the following scope boundaries that contextualise our results.

Experimental scale. The $\mathcal{O}(9^c)$ growth of subexperiment counts restricts practical experimentation to small qubit counts when cutting is embedded in an iterative training loop (Section 6.1). Single-shot cutting toolflows such as CutQC target much larger circuits [10]; our scope limit stems from the multiplicative interaction of cutting overhead with the training-query volume, not from cutting per se. The structural findings—reconstruction dominance, scaling saturation, and straggler sensitivity patterns—are architectural properties of the pipeline that become more pronounced at larger scales, since the reconstruction share increases monotonically with cut count (Table 3).

Runtime and evaluation scope. Our thread-based worker pool isolates pipeline effects from network confounders; the qualitative findings are structural properties rather than threading artefacts. The number of independent runs per configuration (notably $n = 5$ at 3 cuts) limits statistical power; we report medians and 95th percentiles accordingly. The observed trends are directionally consistent across all configurations and both datasets.

8.3. Future work

Minimising reconstruction overhead. The most pressing direction is reducing the reconstruction bottleneck. Promising approaches include approximate reconstruction via tensor-network contractions or low-rank methods, distributed tree-reduction to parallelise the reconstruction phase, incremental reconstruction overlapping with late execution, and integration of reduced-overhead cutting schemes [11, 13] that reduce the base of the exponential from 9 to smaller constants.

Scheduling and extended evaluation. A systematic comparison of batched and staggered dispatch strategies, variance-aware schedulers, and adaptive shot allocation would clarify how scheduling policies interact with the reconstruction barrier. Validating the key findings on a distributed runtime (e.g., Ray [31]), with additional datasets, ansatz depths, multi-class problems, and detailed convergence analysis would strengthen generalisability.

9. Conclusions

We proposed *DistributedEstimator*, a cut-aware estimator execution pipeline for training quantum neural networks via circuit cutting, and evaluated it as a staged distributed workload with explicit measurement of overheads, bottlenecks, and learning outcomes.

Our results demonstrate that circuit cutting introduces substantial end-to-end overheads that grow with cut count (RQ1). Classical reconstruction is a primary contributor to wall-clock time, reaching a median of 53% and a 95th percentile of 58% of per-query estimator time at three cuts, thereby limiting achievable speed-up under increased parallelism (RQ2). Straggler-induced tail effects measurably impact time-to-solution, with sensitivity governed by the balance between execution and reconstruction on the critical path (RQ3). Despite these systems costs, test accuracy is fully preserved on Iris and maintained without systematic degradation on MNIST across all evaluated cut configurations (RQ4). Robustness under Gaussian noise and FGSM perturbations is similarly preserved, with several cut configurations exhibiting comparable or improved robustness relative to the uncut baseline (RQ5).

The exponential growth of subexperiment counts with each additional cut ($\mathcal{O}(9^c)$ for CNOT-based wire cutting) represents a fundamental computational barrier that constrains practical experimentation to small qubit counts with current methods. A targeted validation on the *ibm_kingston* superconducting QPU confirms that reconstruction from physically measured subexperiment outcomes preserves both accuracy and robustness, while revealing that the stage balance inverts on real hardware: execution becomes the dominant cost due to per-circuit device time and queueing, whereas reconstruction stays small in absolute terms. The central architectural conclusion is unchanged under either regime, because the $\mathcal{O}(9^c)$ growth inflates both stages—reconstruction is a purely classical cost that better hardware cannot reduce, and execution latency is a hardware cost that reconstruction optimisation cannot reduce.

This motivates a focused effort on overhead reduction—through approximate reconstruction, distributed reduction, and reduced-overhead cutting schemes—as a prerequisite for scaling circuit cutting to practically relevant QML workloads. Overall, our study establishes that scalable circuit cutting for learning workloads requires treating reconstruction and aggregation as first-class systems problems, and provides a reproducible foundation for designing the scheduling, reconstruction, and variance-aware execution strategies needed to make cutting practical at scale. In addition to the future directions discussed previously, we will explore the integration of our *DistributedEstimator* in quantum application deployment and management frameworks such as QFaaS [49].

CRedit authorship contribution statement

Prabhjot Singh: Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Adel N. Toosi:** Writing – review & editing, Supervision, Conceptualization. **Rajkumar Buyya:** Writing – review & editing, Validation, Supervision, Methodology, Formal analysis, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

- [1] J. Preskill, Quantum computing in the NISQ era and beyond, *Quantum* 2 (2018) 79, <http://dx.doi.org/10.22331/q-2018-08-06-79>.
- [2] M. Cerezo, A. Arrasmith, R. Babbush, S.C. Benjamin, S. Endo, K. Fujii, J.R. McClean, K. Mitarai, X. Yuan, L. Cincio, P.J. Coles, Variational quantum algorithms, *Nat. Rev. Phys.* 3 (9) (2021) 625–644, <http://dx.doi.org/10.1038/s42254-021-00348-9>.
- [3] S. Endo, Z. Cai, S.C. Benjamin, X. Yuan, Hybrid quantum-classical algorithms and quantum error mitigation, *J. Phys. Soc. Japan* 90 (3) (2021) 032001, <http://dx.doi.org/10.7566/JPSJ.90.032001>.
- [4] M. Schuld, V. Bergholm, C. Gogolin, J. Izaac, N. Killoran, Evaluating analytic gradients on quantum hardware, *Phys. Rev. A* 99 (3) (2019) 032331, <http://dx.doi.org/10.1103/PhysRevA.99.032331>.
- [5] K. Temme, S. Bravyi, J.M. Gambetta, Error mitigation for short-depth quantum circuits, *Phys. Rev. Lett.* 119 (18) (2017) 180509, <http://dx.doi.org/10.1103/PhysRevLett.119.180509>.
- [6] J. Dean, S. Ghemawat, MapReduce: Simplified data processing on large clusters, in: *Proceedings of the 6th USENIX Symposium on Operating Systems Design and Implementation, OSDI, 2004*, pp. 137–150.
- [7] M. Zaharia, A. Konwinski, A.D. Joseph, R. Katz, I. Stoica, Improving MapReduce performance in heterogeneous environments, in: *Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation, OSDI, 2008*.
- [8] A. Sergeev, M. Del Balso, Horovod: fast and easy distributed deep learning in TensorFlow, 2018, <http://dx.doi.org/10.48550/arXiv.1802.05799>, arXiv preprint arXiv:1802.05799.
- [9] T. Peng, A.W. Harrow, M. Ozols, X. Wu, Simulating large quantum circuits on a small quantum computer, *Phys. Rev. Lett.* 125 (15) (2020) 150504, <http://dx.doi.org/10.1103/PhysRevLett.125.150504>.
- [10] W. Tang, T. Tomes, M. Suchara, J. Larson, M. Martonosi, CutQC: Using small quantum computers for large quantum circuit evaluations, in: *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS, 2021*, pp. 473–486, <http://dx.doi.org/10.1145/3445814.3446758>.
- [11] C. Piveteau, D. Sutter, Circuit knitting with classical communication, *IEEE Trans. Inform. Theory* 70 (5) (2024) 3001–3016, <http://dx.doi.org/10.1109/TIT.2023.3310797>, Preprint: arXiv:2205.00016.
- [12] A.W. Harrow, A. Lowe, Optimal quantum circuit cuts with application to clustered Hamiltonian simulation, *PRX Quantum* 6 (1) (2025) 010316, <http://dx.doi.org/10.1103/PRXQuantum.6.010316>.
- [13] A. Lowe, M. Medvidović, A. Hayes, L.J. Romero, N.M. Tubman, D. Camps, Fast quantum circuit cutting with randomized measurements, *Quantum* 7 (2023) 934, <http://dx.doi.org/10.22331/q-2023-03-02-934>.
- [14] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, A. Vladu, Towards deep learning models resistant to adversarial attacks, in: *International Conference on Learning Representations, ICLR, 2018*.
- [15] F. Tramèr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, P. McDaniel, Ensemble adversarial training: Attacks and defenses, in: *International Conference on Learning Representations, ICLR, 2018*.
- [16] Qiskit Machine Learning Contributors, Qiskit machine learning, 2024, <https://qiskit.org/ecosystem/machine-learning/>. (Accessed April 2026).
- [17] Qiskit Addon Cutting Contributors, Circuit cutting with qiskit addons, 2024, <https://qiskit.github.io/qiskit-addon-cutting/>. (Accessed April 2026).
- [18] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P.J. Love, A. Aspuru-Guzik, J.L. O'Brien, A variational eigenvalue solver on a photonic quantum processor, *Nat. Commun.* 5 (2014) 4213, <http://dx.doi.org/10.1038/ncomms5213>.
- [19] N. Moll, P. Barkoutsos, L.S. Bishop, J.M. Chow, A. Cross, D.J. Egger, S. Filipp, A. Fuhrer, J.M. Gambetta, M. Ganzhorn, A. Kandala, et al., Quantum optimization using variational algorithms on near-term quantum devices, *Quantum Sci. Technol.* 3 (3) (2018) 030503, <http://dx.doi.org/10.1088/2058-9565/aab822>.
- [20] A. Kandala, A. Mezzacapo, K. Temme, M. Takita, M. Brink, J.M. Chow, J.M. Gambetta, Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets, *Nature* 549 (7671) (2017) 242–246, <http://dx.doi.org/10.1038/nature23879>.
- [21] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, S. Lloyd, Quantum machine learning, *Nature* 549 (7671) (2017) 195–202, <http://dx.doi.org/10.1038/nature23474>.
- [22] K. Mitarai, M. Negoro, M. Kitagawa, K. Fujii, Quantum circuit learning, *Phys. Rev. A* 98 (3) (2018) 032309, <http://dx.doi.org/10.1103/PhysRevA.98.032309>.
- [23] V. Havlíček, A.D. Córcoles, K. Temme, A.W. Harrow, A. Kandala, J.M. Chow, J.M. Gambetta, Supervised learning with quantum-enhanced feature spaces, *Nature* 567 (7747) (2019) 209–212, <http://dx.doi.org/10.1038/s41586-019-0980-2>.
- [24] H.-Y. Huang, M. Broughton, M. Mohseni, R. Babbush, S. Boixo, H. Neven, J.R. McClean, Power of data in quantum machine learning, *Nat. Commun.* 12 (2021) 2631, <http://dx.doi.org/10.1038/s41467-021-22539-9>.
- [25] J.R. McClean, S. Boixo, V.N. Smelyanskiy, R. Babbush, H. Neven, Barren plateaus in quantum neural network training landscapes, *Nat. Commun.* 9 (2018) 4812, <http://dx.doi.org/10.1038/s41467-018-07090-4>.

- [26] A. Javadi-Abhari, M. Treinish, K. Krsulich, C.J. Wood, J. Lishman, J. Gacon, S. Martiel, P.D. Nation, L.S. Bishop, A.W. Cross, B.R. Johnson, J.M. Gambetta, Quantum computing with qiskit, 2024, <http://dx.doi.org/10.48550/arXiv.2405.08810>, arXiv preprint [arXiv:2405.08810](https://arxiv.org/abs/2405.08810).
- [27] I.B.M. Quantum, Qiskit primitives API documentation (estimator and sampler), 2024, URL <https://quantum.cloud.ibm.com/docs/api/qiskit/primitives>. (Accessed April 2026). Online documentation.
- [28] V. Bergholm, J. Izaac, M. Schuld, C. Gogolin, N. Killoran, et al., PennyLane: Automatic differentiation of hybrid quantum-classical computations, 2018, <http://dx.doi.org/10.48550/arXiv.1811.04968>, arXiv preprint [arXiv:1811.04968](https://arxiv.org/abs/1811.04968).
- [29] M. Broughton, G. Verdon, T. McCourt, A.J. Martinez, J.H. Yoo, S.V. Isakov, P. Massey, R. Halavati, M.Y. Niu, A. Zlokapa, et al., TensorFlow Quantum: A software framework for quantum machine learning, 2020, <http://dx.doi.org/10.48550/arXiv.2003.02989>, arXiv preprint [arXiv:2003.02989](https://arxiv.org/abs/2003.02989).
- [30] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M.J. Franklin, S. Shenker, I. Stoica, Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing, in: Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI, 2012.
- [31] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul, M.I. Jordan, I. Stoica, Ray: A distributed framework for emerging AI applications, in: Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation, OSDI, 2018.
- [32] M. Li, D.G. Andersen, J.W. Park, A.J. Smola, A. Ahmed, V. Josifovski, J. Long, E.J. Shekita, B.-Y. Su, Scaling distributed machine learning with the parameter server, in: Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation, OSDI, 2014, pp. 583–598.
- [33] Q. Ho, J. Cipar, H. Cui, J.K. Kim, S. Lee, P.B. Gibbons, G. Gibson, G.R. Ganger, E.P. Xing, More effective distributed ML via a stale synchronous parallel parameter server, in: Advances in Neural Information Processing Systems, NeurIPS, 2013.
- [34] B. Recht, C. Re, S. Wright, F. Niu, Hogwild!: A lock-free approach to parallelizing stochastic gradient descent, in: Advances in Neural Information Processing Systems, NeurIPS, 2011.
- [35] M. Cho, et al., BlueConnect: Decomposing all-reduce for deep learning on heterogeneous network hierarchy, in: Proceedings of Machine Learning and Systems, (MLSys), 2019.
- [36] P. Goyal, P. Dollár, R.B. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, K. He, Accurate, large minibatch SGD: Training ImageNet in 1 hour, 2017, <http://dx.doi.org/10.48550/arXiv.1706.02677>, arXiv preprint [arXiv:1706.02677](https://arxiv.org/abs/1706.02677).
- [37] R.D. Blumofe, C.E. Leiserson, Scheduling multithreaded computations by work stealing, J. ACM 46 (5) (1999) 720–748, <http://dx.doi.org/10.1145/324133.324234>.
- [38] A. Carrera Vazquez, C. Tornow, D. Ristè, S. Woerner, M. Takita, D.J. Egger, Combining quantum processors with real-time classical communication, Nature 636 (2024) 75–79, <http://dx.doi.org/10.1038/s41586-024-08178-2>.
- [39] J. Garrison, et al., Circuit cutting with quantum serverless, 2023, URL <https://research.ibm.com/publications/circuit-cutting-with-quantum-serverless>. (Accessed April 2026). IBM Research.
- [40] I. Sitdikov, et al., Circuit knitting toolbox and quantum serverless, 2023, URL <https://research.ibm.com/publications/circuit-knitting-toolbox-and-quantum-serverless>. (Accessed April 2026). IBM Research.
- [41] C. Johnson, et al., Simulating larger quantum circuits with circuit cutting and quantum serverless, 2023, URL <https://research.ibm.com/publications/simulating-larger-quantum-circuits-with-circuit-cutting-and-quantum-serverless>. (Accessed April 2026). SC 2023 Poster, IBM Research.
- [42] Y.J. Patel, A. Kundu, M. Ostaszewski, X. Bonet-Monroig, V. Dunjko, O. Danaci, Curriculum reinforcement learning for quantum architecture search under hardware errors, in: International Conference on Learning Representations, ICLR, 2024, <http://dx.doi.org/10.48550/arXiv.2402.03500>, Preprint: [arXiv:2402.03500](https://arxiv.org/abs/2402.03500).
- [43] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, R. Fergus, Intriguing properties of neural networks, in: International Conference on Learning Representations, ICLR, 2014, Preprint: [arXiv:1312.6199](https://arxiv.org/abs/1312.6199).
- [44] I.J. Goodfellow, J. Shlens, C. Szegedy, Explaining and harnessing adversarial examples, in: International Conference on Learning Representations, ICLR, 2015.
- [45] Qiskit Contributors, Qiskit: An open-source framework for quantum computing, 2024, <https://qiskit.org>. (Accessed April 2026).
- [46] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86 (11) (1998) 2278–2324, <http://dx.doi.org/10.1109/5.726791>.
- [47] R.A. Fisher, The use of multiple measurements in taxonomic problems, Ann. Eugen. 7 (2) (1936) 179–188, <http://dx.doi.org/10.1111/j.1469-1809.1936.tb02137.x>.
- [48] G.M. Amdahl, Validity of the single processor approach to achieving large scale computing capabilities, in: Proceedings of the April 18–20, 1967, Spring Joint Computer Conference, AFIPS, 1967, pp. 483–485, <http://dx.doi.org/10.1145/1465482.1465560>.
- [49] H.T. Nguyen, M. Usman, R. Buyya, QFaaS: A serverless function-as-a-service framework for quantum computing, Future Gener. Comput. Syst. 154 (2024) 281–300, <http://dx.doi.org/10.1016/j.future.2024.01.018>.