QCLINK: Offloading Hybrid Quantum-Classical Computation to SmartNICs via RDMA

Xingdong Li[1*], Yongzhuo Lu[1*], Zhipeng Liu[1], Xiaofei Yue[2],
Junyu Chen[1], Yu Peng[1], Tingting Li[1†], Zhaoxuan Li[3], Ziming Zhao[1†], Jianwei Yin[1†]
[1]School of Software Technology, Zhejiang University, China
[2]School of Computer Science and Technology, Beijing Institute of Technology, China
[3]Institute of Information Engineering, Chinese Academy of Sciences, China
Emails: {litt2020, zhaoziming}@zju.edu.cn, zjuyjw@cs.zju.edu.cn

***Abstract*—Hybrid quantum-classical algorithms rely on tight iterative control loops that alternate between quantum evaluations and classical parameter updates. Despite recent efforts on accelerating quantum execution and real-time control paths, the end-to-end performance of these workloads is often bottlenecked by the distributed control plane, where iterations incur frequent fine-grained message exchanges, host CPU mediation, and amplified tail latency. In this paper, we take a networking perspective on hybrid quantum-classical computing and show that the classical optimization loop constitutes a key scalability barrier. We present QCLINK (Quantum-Classical Link), a SmartNIC-assisted offloading framework that moves hybrid optimization into the network data plane and leverages RDMA to enable low-latency, CPU-bypassed communication between classical controllers and quantum backends. By executing parameter perturbation, gradient estimation, and iteration control directly on the SmartNIC, QCLINK turns hybrid optimization into a data-plane service, reducing per-iteration overhead and mitigating control-loop jitter. We implement a prototype of QCLINK with SmartNIC-resident optimization logic and RDMA-based messaging primitives. Experiments demonstrate that QCLINK significantly improves per-iteration latency and end-to-end completion time over host-centric baselines, while stabilizing tail latency and freeing host CPU resources under concurrent workloads.**

***Index Terms*—Quantum-Classical Computation, SmartNIC, RDMA, Computation Offloading**

I. INTRODUCTION

Hybrid quantum-classical algorithms are a practical approach for near-term quantum computing on noisy intermediate-scale quantum (NISQ) devices [38]. Representative examples include variational quantum algorithms [4], [34], [7] and stochastic optimization methods such as the Simultaneous Perturbation Stochastic Approximation (SPSA) [39], [20], which iteratively alternate between quantum evaluations and classical parameter updates. In these algorithms, quantum circuits are repeatedly executed to evaluate objective functions, while classical optimizers adjust parameters based on measurement outcomes, forming a tight feedback loop across heterogeneous computing components [10], [11].

Despite significant advances in quantum hardware and execution frameworks [40], the end-to-end performance of hybrid quantum-classical algorithms remains far from optimal.

*Equal contribution and †corresponding authors.

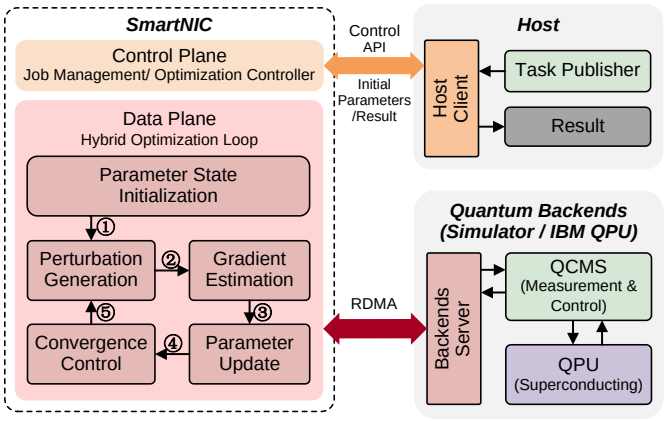

Fig. 1. QCLINK overview. The SmartNIC executes and dispatches backend evaluations over RDMA, removing per-iteration host involvement.

Recent system efforts have largely focused on accelerating quantum execution or improving real-time control paths between classical hosts and quantum devices [11], [30]. However, in practical hybrid workloads, overall completion time is often dominated not by quantum execution itself, but by the distributed control loop that orchestrates frequent, fine-grained interactions between classical optimizers and quantum backends. This control loop typically involves high-frequency, small-message exchanges, repeated synchronization, and substantial CPU involvement, making it highly sensitive to network latency, software overheads, and tail-latency variability [5], [45].

From a distributed systems perspective, hybrid quantum-classical algorithms expose a fundamental mismatch between their iterative control structure and conventional host-centric execution models. In common deployments, the classical optimization loop is executed on a host CPU, while each quantum evaluation is issued as a remote procedure call to a quantum backend or simulator. Even when high-performance interconnects are available, the repeated traversal of control and data planes, through protocol stacks, context switches, and serialization layers, introduces significant overheads that scale poorly with iteration count and concurrency [45], [54]. As a result, quantum resources may remain underutilized while classical control becomes the system bottleneck [30], [11].

We consider that the classical optimization loop of hybrid algorithms should be treated as a first-class target for systems-level acceleration. We observe that the control logic of many hybrid algorithms, including SPSA, exhibits simple and structured computation patterns: parameter perturbation, objective aggregation, gradient estimation, and convergence control [39], [20]. These operations are lightweight yet latency-critical, and are executed repeatedly along the critical path of the algorithm. This makes them well-suited for offloading to programmable network devices that can operate at low latency and bypass host CPUs [2], [15], [28], [54].

To this end, we propose a SmartNIC-based framework that offloads hybrid quantum-classical optimization loops to the network data plane and leverages RDMA for efficient communication with quantum backends (Figure 1). Our design migrates the per-iteration control logic of hybrid algorithms, including parameter management, perturbation generation, and update decisions, onto the SmartNIC, while using RDMA to directly exchange evaluation results with quantum execution engines. By transforming the hybrid optimization loop into a data-plane service, our approach eliminates repeated host involvement from the critical path, reduces control-loop latency, and improves scalability under concurrent workloads.

We implement a prototype system that supports SPSA-style hybrid algorithms and integrates SmartNIC-resident optimization logic with RDMA-based communication. The system exposes a lightweight control-plane interface for job submission and monitoring, while executing the entire optimization loop in the data plane. Through experimental evaluation, we demonstrate that our approach significantly reduces per-iteration latency and end-to-end completion time compared to host-centric baselines, while also improving CPU efficiency and tail-latency stability. These results highlight the benefits of rethinking hybrid quantum-classical computing as a distributed systems problem and exploiting in-network acceleration to address its control-plane bottlenecks.

In summary, this paper makes the following contributions.

- We identify the classical optimization loop as a primary performance bottleneck in hybrid quantum-classical algorithms from a distributed systems perspective.
- We propose a SmartNIC-based offloading framework that transforms hybrid optimization loops into data-plane services using RDMA-enabled communication.
- We implement QCLINK and demonstrate substantial improvements in latency, completion time, and scalability compared to host-centric baselines.

II. BACKGROUND AND MOTIVATION

A. Hybrid Quantum-Classical Optimization Loops

Hybrid quantum-classical algorithms are a practical approach for leveraging near-term quantum devices by combining quantum evaluations with classical optimization [38], [4]. A typical execution model consists of an *outer* classical loop that iteratively updates a parameter vector and an *inner* quantum loop that evaluates an objective function by executing parameterized quantum circuits and aggregating measurement outcomes [34], [7]. This structure forms a closed feedback loop in which each parameter update depends on the most recent quantum evaluation results [30], [11].

TABLE I
CHARACTERISTIC MATRIX OF REPRESENTATIVE WORKLOADS.

Workload	Light compute	Serialized loop	Small & frequent	Ctrl-plane dom.
Hybrid QC opt.	✓	✓	✓	✓
ML training	×	×	×	×
HPC collectives	×	×	×	×
QC control	✓	✓	✓	✓

Among hybrid optimizers, Simultaneous Perturbation Stochastic Approximation (SPSA) is widely used due to its robustness to noise and low evaluation complexity per iteration [39], [20]. In iteration k, SPSA samples a random perturbation direction and performs multiple objective evaluations (*e.g.,* at $\theta_k \pm \delta_k$) to estimate a stochastic gradient, which is then used to update parameters before proceeding to iteration $k{+}1$ [39]. Because updates cannot proceed without the returned measurement outcomes, the optimization loop is sequential and latency-sensitive [30], [11].

From a systems perspective, hybrid optimization loops exhibit a distinct workload signature: (i) *light compute*: per-iteration classical computation (perturbation, aggregation, update rules) is lightweight; (ii) *serialized loop*: iterations are strongly dependent and offer limited opportunities to hide latency via parallelism; (iii) *small & frequent messages*: the control loop repeatedly exchanges short request/result messages at a high rate [19]. Consequently, end-to-end completion time is often driven more by orchestration overhead and its tail variability than by raw compute throughput [5], [37], [33].

A convenient abstraction is:

$$T_{\text{iter}} \approx N_{\text{eval}} \cdot (T_{\text{backend}} + T_{\text{net}} + T_{\text{ctrl}}) + T_{\text{upd}}, \qquad (1)$$

where N_{eval} is the number of quantum evaluations per iteration, T_{backend} is backend execution time, T_{net} is transport time, T_{ctrl} is control-plane overhead (*e.g.,* software stack, scheduling, orchestration, and buffer management), and T_{upd} is the parameter update time [11], [30]. Given the serialized dependency, even modest T_{ctrl} and its tail behavior are repeatedly amplified across iterations, directly impacting completion time [5], [18]. Table I summarizes representative workloads along *Light compute*, *Serialized loop*, *Small & frequent*, and *Control-plane dominated*, highlighting why hybrid optimization warrants dedicated system support beyond throughput-oriented designs.

B. Limitations of Host-Centric Execution Models

Most existing quantum software stacks execute hybrid algorithms in a host-centric manner: the classical optimization loop runs on a host CPU, while quantum evaluations are dispatched to a remote backend or simulator through a layered software stack (*e.g.,* runtime scheduler, serialization, networking libraries, and RPC frameworks)[11], [30], [19]. Measurement results are returned to the host, which computes the next parameters and issues subsequent evaluation requests [11], [30]. While

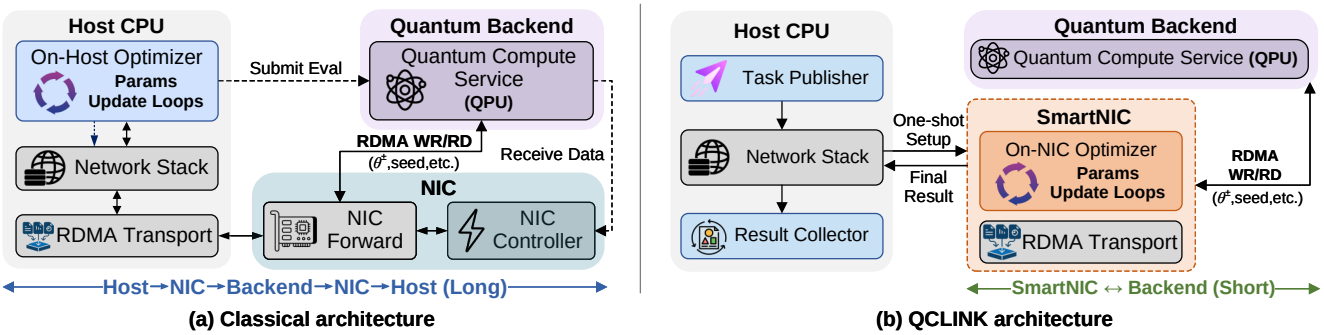

Fig. 2. System architecture comparison. QCLINK offloads the hybrid optimization loop to the SmartNIC and uses RDMA for backend dispatch, eliminating host involvement from the per-iteration critical path.

straightforward, this architecture is a poor fit for the workload signature in Table I, since it keeps the host on the per-iteration critical path and repeatedly incurs control-plane overheads [19], [54]. We highlight three fundamental limitations.

Orchestration Overhead on the Critical Path. Each iteration requires multiple request-response interactions between the host and the backend. These interactions traverse general-purpose protocol stacks and CPU-mediated paths, and the resulting per-evaluation overhead is serialized by the optimization dependency [19], [54].

CPU Mediation Persists. Even when high-performance transports such as RDMA are employed, the host often remains responsible for orchestration, buffer management, and control decisions [6], [44], [22]. RDMA can reduce T_{net} (see Eq. 1), but it does not structurally eliminate T_{ctrl}, which frequently dominates the critical path for small, frequent control exchanges [19], [54].

Tail-Latency Amplification under Contention. Host-centric execution is vulnerable to OS scheduling jitter, interrupts, and contention among concurrent jobs [37], [33], [18]. Since iterations are serialized, tail events in T_{ctrl} directly inflate T_{iter}, degrading completion time and potentially underutilizing scarce quantum resources in multi-tenant settings [5]. Several recent systems advocate tighter coupling between classical and quantum components using high-speed interconnects and accelerator-aware runtimes, often targeting real-time data paths (*e.g.,* low-latency quantum control and error correction) [10], [11]. However, these efforts do not fundamentally restructure the hybrid optimization *control loop*: repeated control-plane interactions and host-mediated orchestration remain on the critical path [30].

C. Motivation for Data-Plane Offloading

Given the signature in Table I, a serialized loop with lightweight compute and small, frequent messages, the key optimization opportunity is to reduce control-plane overhead and its variability on the per-iteration critical path [5], [37], [33]. This motivates rethinking *where* the optimization loop executes, rather than only improving transport latency. Modern SmartNICs provide programmable execution environments with direct access to high-speed networking primitives (*e.g.,* RDMA, shared memory, and event-driven processing) [22], [54]. By relocating the hybrid optimization loop onto a SmartNIC, we can remove the host CPU from the per-iteration critical path: the SmartNIC can directly exchange parameter updates and evaluation results with the quantum execution engine via RDMA, bypassing host protocol stacks and minimizing control-plane overhead in Eq. 1 [54], [45]. Equally important, executing latency-sensitive control decisions in a data-plane environment mitigates host-induced tail variability stemming from OS scheduling and interference [37], [33], [18].

Conceptually, data-plane offloading transforms hybrid optimization from a host-driven RPC workflow into a *network-resident state machine service* [17]. Instead of repeatedly invoking remote procedures from the host, the SmartNIC autonomously maintains iteration state, issues evaluation requests, and applies update rules upon receiving results [54], [45], [2]. The host interacts with the system only at coarse granularity (job submission, monitoring, and final result retrieval), improving scalability and predictability [16], [17].

III. QCLINK DESIGN

This section presents the design of QCLINK, focusing on its architecture and data-plane offloading mechanisms.

A. Overview

The design of QCLINK follows three goals: (i) minimizing end-to-end latency by removing the host from the per-iteration critical path [5], [33]; (ii) supporting common hybrid optimizers such as SPSA without backend-specific assumptions [39]; and (iii) scaling to concurrent jobs with predictable performance [14], [45]. We observe that many hybrid workloads have lightweight per-iteration control logic but strict iteration dependencies, making orchestration overhead and its tail behavior a dominant bottleneck [5], [19].

To address this bottleneck, QCLINK offloads the hybrid optimization loop to a programmable SmartNIC and leverages RDMA for low-latency, CPU-bypassed communication [6], [44] with quantum execution backends. The system consists of three components: (i) a host-side client for coarse-grained job submission and result retrieval, (ii) a SmartNIC-resident optimization service that executes latency-critical per-iteration logic in the network data plane [16], [17], [15], and (iii) one or more quantum execution backends that run parameterized circuits

and expose RDMA-accessible memory regions [6], [44]. Each job is represented on the SmartNIC as a state machine [17], [26], encapsulating the parameter vector, perturbations, step-size schedules, iteration counters, and convergence state; the SmartNIC schedules evaluations, processes returned statistics, and decides subsequent actions autonomously.

Figure 2 contrasts the conventional host-centric execution model with QCLINK. In the classical architecture (left), the optimizer runs on the host CPU, and each iteration traverses a long control path: the host constructs and submits evaluation requests, the kernel network stack and protocol processing serialize small messages [19], [35], and results return to the host before the next update can proceed. This design introduces per-iteration control-plane overheads that accumulate linearly with the iteration count and are highly sensitive to software jitter, context switching, and tail-latency variability. In contrast, QCLINK (right) shortens the critical path by relocating the optimization loop onto the SmartNIC service layer. The host interacts through a coarse-grained control-plane interface (one-shot job submission and final result retrieval), while the SmartNIC performs all per-iteration operations locally and exchanges circuit parameters and evaluation results with the backend via RDMA write/read operations (WR/RD), bypassing host networking stacks and CPU mediation. As a result, the per-iteration critical path becomes a short SmartNIC-to-backend datapath, improving both iteration latency and stability under repeated evaluations.

B. Control Plane and Data Plane Separation

A key design principle of our system is the explicit separation between control-plane and data-plane responsibilities. This separation is intentional and central to eliminating distributed control overheads in hybrid optimization workloads. The control plane provides a stable interface for job admission, configuration, and lifecycle management [35]. Control-plane operations occur at coarse granularity and are not latency sensitive. Hosts interact with the system through a lightweight API to submit jobs, monitor progress, and retrieve final results.

In contrast, the data plane executes the hybrid optimization loop and handles all per-iteration operations. These include generating parameter perturbations, issuing quantum evaluation requests, receiving results, computing updates, and checking convergence conditions. By confining these operations to the data plane, the system removes repeated host involvement from the critical path and significantly reduces sensitivity to operating system scheduling and interrupt variability. This separation allows the SmartNIC to operate autonomously during optimization, effectively transforming hybrid quantum-classical execution into a network-resident service [17] rather than a host-driven workflow.

C. SmartNIC-Resident Optimization Loop

The SmartNIC executes the core logic of hybrid optimization algorithms as a resident data-plane service. We model the optimization process as a state machine that advances upon the completion of quantum evaluations or timeout events.

Algorithm 1 SmartNIC-Resident SPSA Optimization Loop

```
Initialize θ_0, iteration counter k ← 0
while not converged do
    Generate random perturbation Δ_k
    Compute θ_k^+, θ_k^-
    Issue quantum evaluations for θ_k^+ and θ_k^-
    Wait for results y_k^+, y_k^-
    Compute gradient estimate ĝ_k
    Update parameters θ_{k+1}
    k ← k + 1
end while
Return final parameters θ_k
```

For instance, we elaborate on the Simultaneous Perturbation Stochastic Approximation (SPSA) algorithm [39], [20] (Algorithm 1). Let $\boldsymbol{\theta}_k$ denote the parameter vector at iteration k. SPSA generates a random perturbation vector $\boldsymbol{\Delta}_k$, where each element is independently drawn from a symmetric Bernoulli distribution. Two quantum evaluations are performed at perturbed parameter points:

$$\boldsymbol{\theta}_k^+ = \boldsymbol{\theta}_k + c_k \boldsymbol{\Delta}_k, \quad \boldsymbol{\theta}_k^- = \boldsymbol{\theta}_k - c_k \boldsymbol{\Delta}_k, \tag{2}$$

where c_k is a perturbation scale.

Let y_k^+ and y_k^- denote the objective values returned by the quantum backend. The gradient estimate is computed as:

$$\hat{\boldsymbol{g}}_k = \frac{y_k^+ - y_k^-}{2c_k} \boldsymbol{\Delta}_k^{-1}, \tag{3}$$

where $\boldsymbol{\Delta}_k^{-1}$ denotes element-wise inversion. The parameter update rule is:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - a_k \hat{\boldsymbol{g}}_k, \tag{4}$$

with step size a_k. All operations above are executed on the SmartNIC without host involvement. The SmartNIC also evaluates convergence conditions, such as maximum iteration count or objective thresholds, to determine termination. To support multiple concurrent jobs, the SmartNIC maintains independent state machines and scheduling queues. As shown in Algorithm 1, each optimization job advances as an event-driven state machine on the SmartNIC, enabling efficient multiplexing across concurrent jobs.

D. Data-Plane Communication Interface

Communication between the SmartNIC and quantum execution backends is implemented as a data-plane interface using RDMA [6], [44]. For each job, memory regions are registered and shared between the SmartNIC and the backend to exchange parameter updates and evaluation results. The SmartNIC writes perturbed parameters directly into backend-accessible memory, triggering quantum circuit execution. After execution, the backend writes aggregated objective values or measurement statistics back into SmartNIC-accessible memory. Result availability is detected using event-driven mechanisms, allowing the SmartNIC to immediately proceed with parameter updates. This RDMA-based interface eliminates data copies

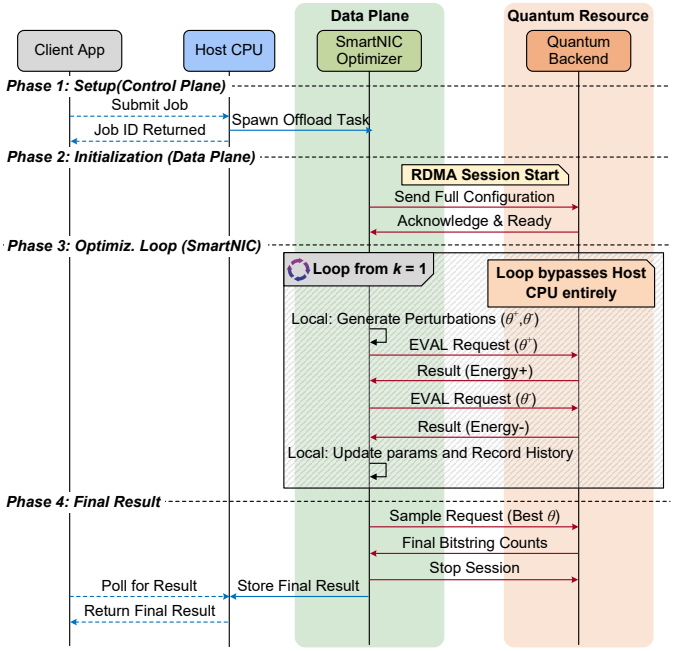

Fig. 3. Sequence diagram of QCLINK. QCLINK offloads the iterative optimization loop to the SmartNIC and uses RDMA for backend-facing evaluations, eliminating host involvement from the per-iteration critical path.

and avoids kernel networking stacks [6], [44], [35], providing low-latency and predictable communication. The abstraction decouples optimization logic from backend implementation details, enabling support for simulators and hardware quantum devices that expose RDMA-accessible interfaces.

E. Concurrency and Scheduling Handling

The design explicitly considers concurrency and robustness in distributed environments. The SmartNIC scheduler enforces per-job resource limits and prevents individual jobs from monopolizing data-plane execution [14], [45]. By interleaving state-machine execution across jobs, the system supports concurrent optimization workloads with predictable performance [29]. Fault handling is implemented at the data-plane level. Each quantum evaluation is associated with a configurable timeout [19], [6]. If results are delayed or unavailable, the SmartNIC can reissue evaluation requests or terminate the job according to policy. Transient communication failures are handled transparently by RDMA mechanisms, while persistent backend failures are reported to the host through the control plane. These mechanisms ensure that the system remains robust under backend variability and network disruptions, further improving the reliability of hybrid quantum-classical execution.

IV. IMPLEMENTATION

We implement a prototype of QCLINK to validate the feasibility and benefits of offloading evaluation-intensive hybrid quantum-classical workloads to the data plane [54], [26]. The implementation is modular: a generic *job substrate* on the SmartNIC provides (i) per-job state management, (ii) a persistent backend session, and (iii) a request/response dispatch loop, while workload-specific control logic (*e.g.,* SPSA) is implemented as a pluggable module [21], [14]. In this prototype, we instantiate the substrate with an SPSA-based variational optimization loop [39], [4].

Execution Model. Figure 3 summarizes the workflow as a coarse-grained job submission path plus a latency-critical on-NIC iteration path. The host submits a job and receives a job identifier; the SmartNIC then establishes a backend session and transfers the immutable job configuration (*e.g.,* circuit template, optimizer hyperparameters, termination criteria). After initialization, the SmartNIC executes the optimization loop autonomously: it generates per-iteration evaluation descriptors (*e.g.,* θ^+/θ^- for SPSA), issues backend evaluations, waits for results, and updates the job state without host involvement. Upon termination, the SmartNIC performs an optional final sampling with the best parameters, stores the result, and exposes it to the host for retrieval (Figure 4). This organization turns a fine-grained host-driven loop into a job-style control-plane interaction, keeping repeated backend exchanges on the SmartNIC-to-backend datapath.

SmartNIC-Resident Service. The optimization service is implemented as a long-running, event-driven execution engine that maintains on-device state for active jobs [26]. Each job is a compact state machine containing: (i) workload state (*e.g.,* parameter vector and circuit descriptors), (ii) optimizer metadata (*e.g.,* SPSA step-size and perturbation schedules), (iii) counters, and termination conditions. To minimize latency jitter, the service uses polling to advance job state immediately upon result availability or timeout events [37], [33]. The SPSA arithmetic (perturbation generation, gradient estimate, and vector update) is implemented using fixed-size routines suitable for SmartNIC execution.

RDMA Datapath to Quantum Backends. SmartNIC-backend communication uses RDMA for low-latency, CPU-bypassed transfers [6], [44]. For each job, the SmartNIC creates RDMA queue pairs and registers memory regions (MRs) used to exchange evaluation descriptors and results. The SmartNIC writes descriptors (*e.g.,* parameter payloads or batch identifiers) into backend-accessible MRs; the backend consumes them to enqueue circuit executions and writes objective values or measurement aggregates back into SmartNIC-accessible MRs. The SmartNIC detects completion via MR polling and immediately resumes the optimizer. This design avoids kernel networking stacks and serialization/RPC on the per-iteration path, improving predictability for small, frequent exchanges [35], [19].

Concurrency and Robustness. To support multiple concurrent jobs, the service includes a centralized scheduler that multiplexes state machines across active jobs and enforces fairness using per-job credits (*i.e.,* a bound on outstanding backend evaluations) [14], [45]. Failures are handled with per-request timeouts and simple policies (retry or fail-fast). Transient communication errors rely on RDMA retry mechanisms, while persistent backend failures are surfaced to the host through the control plane [6], [44].

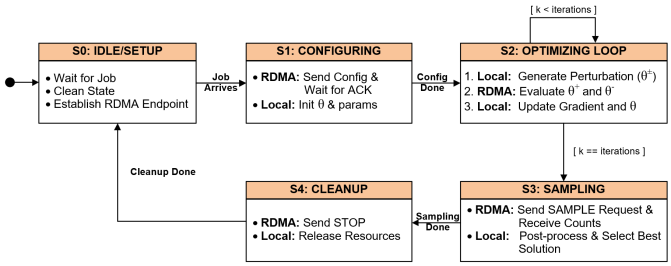

Fig. 4. Execution FSM of the QCLINK offload mechanic. QCLINK represents each job as an on-NIC state machine that repeatedly dispatches circuit evaluations and consumes results to advance the optimizer. The backend-facing path uses RDMA for low-overhead request/response exchange, removing host involvement from the per-iteration critical path.

V. EVALUATION

We conduct a series of evaluations for QCLINK. The experiments are designed to answer the following questions.

- *Latency*: How does moving the backend-facing path from HTTP to RDMA affect per-evaluation latency and per-iteration latency for hybrid optimization workloads (*e.g.,* QAOA with SPSA/COBYLA/GD)?
- *E2E Performance*: How does QCLINK impact end-to-end completion time compared to host-centric baselines?
- *CPU Overhead*: How much client-side CPU overhead is reduced by offloading request orchestration into the SmartNIC service layer?
- *Concurrency and Stability*: How does QCLINK behave under concurrent optimization jobs in terms of throughput and tail latency?

A. Experimental Setup

Testbed and Deployment. Our testbed follows a three-tier deployment consisting of (i) a host client that submits optimization jobs, (ii) a SmartNIC service layer that may execute the optimization loop, and (iii) a quantum backend that evaluates parameterized circuits. The client communicates with the SmartNIC service through an HTTP/REST interface [8], [9]. We deploy the service on an AMD Alveo U280 data center accelerator card (XCU280 UltraScale+ FPGA) integrated into the host via PCIe (Gen3 x16/Gen4 x8) and equipped with 8 GB HBM2, 32 GB DDR4, and dual QSFP28 100 GbE ports [48]. The U280 platform provides a static region that manages PCIe, DMA, and card management, while the optimization logic is implemented in the reconfigurable region and interacts with host software through lightweight control and status interfaces [48], [21]. The SmartNIC service maintains per-job state and orchestrates circuit evaluations; in NIC-Loop baselines, it executes the optimizer loop and autonomously issues evaluation requests to the backend [54]. Unless otherwise specified, we evaluate 12-node MaxCut with QAOA on depth $p = 5$ quantum circuit for 50 iterations, using 512 shots per evaluation and 4096 shots for the final measurement.

Quantum Backends. Since today's quantum cloud providers expose only HTTP/REST-style interfaces (no RDMA), RDMA-based experiments use an RDMA-capable backend emulator. For HTTP-based baselines, we use an IBM Quantum utility-scale system (Eagle-class 127-qubit, heavy-hex). For RDMA-based baselines, we use an RDMA backend emulator configured with IBM-like timing characteristics to isolate transport/control overhead.

Workloads and Optimizers. We evaluated QAOA [7] for MaxCut [13] instances and ran hybrid optimization with multiple classical optimizers, including SPSA [39], COBYLA [36], and gradient descent (GD) [31]. Unless otherwise specified, each optimization run uses a fixed iteration (or evaluation) budget and identical initial parameters across baselines.

Baselines. We implement the host-loop baselines using the open NVQLink framework [32], which provides a reference execution pipeline for hybrid quantum-classical workloads by letting the host repeatedly dispatch circuit-evaluation requests to a backend service. In our experiments, NVQLink serves as a host-centric baseline that exposes per-iteration orchestration overhead and host-side contention effects. We compare our approach against these following baselines, which reflect common execution models for hybrid quantum-classical algorithms:

- *NVQLink (Host-HTTP)* [9]: The client directly issues HTTP requests to the quantum backend for each circuit evaluation, without an intermediate service layer.
- *NVQLink (Host-RDMA)* [6]: Just like HTTP based method above, the client directly issues RDMA requests to the quantum backend for each circuit evaluation, without an intermediate service layer.
- QCLINK *(NIC-HTTP)* [26]. The host submits a job through HTTP; the SmartNIC executes the optimizer loop and communicates with the backend over HTTP.
- QCLINK *(NIC-RDMA)* [54]. The client interacts with the SmartNIC service through HTTP, while the SmartNIC communicates with the backend via RDMA.

We apply the same set of optimizers (SPSA, COBYLA, and GD) to all baselines for a fair comparison.

Metrics. We focus on metrics that capture both end-to-end performance and system-level efficiency:

- *Per-Iteration Latency*: The time required to complete a single optimization iteration, including parameter perturbation, quantum evaluation, and parameter update.
- *Classical Computation(CC) Latency*: The average time spent on classical-side optimizer computation per iteration, measured to quantify the control-plane computation cost and the effectiveness of SmartNIC offloading.
- *Completion Time(CT)*: The total wall-clock time elapsed from the initial job submission to retrieval of final result.
- *Host CPU Utilization*: Average CPU utilization on the host during optimization, measured to assess offloading effectiveness.
- *Tail Latency*: High-percentile (*e.g.,* p99) per-iteration latency, used to quantify execution variability and jitter.
- *Throughput*: The number of optimization jobs completed per unit time as the number of concurrent jobs increases.

TABLE II
LATENCY PERFORMANCE COMPARISON ACROSS ALGORITHMS AND ARCHITECTURES

Algorithm	*Architecture*	*Classical Compute (μs)*	*CC Speedup*	*Per-iteration (ms)*	*P99 (ms)*	*Completion Time (ms)*	*CT Speedup*	*P99 Improv.*
SPSA	NVQLink (HTTP)	262 ± 12	1.00×	38.86 ± 1.35	57.23 ± 6.56	2020.82 ± 67.16	1.00×	-
	NVQLink (RDMA)	284 ± 20	0.92×	35.55 ± 1.00	46.20 ± 2.94	1781.05 ± 43.34	1.13×	1.23×
	QCLINK (HTTP)	142 ± 14	1.84×	10.83 ± 0.37	18.60 ± 5.18	590.42 ± 24.50	3.42×	3.08×
	QCLINK (RDMA)	*133 ± 9*	*1.97×*	*6.47 ± 0.21*	*10.31 ± 2.43*	*339.44 ± 14.03*	*5.95×*	*5.55×*
COBYLA	NVQLink (HTTP)	354 ± 27	1.00×	19.72 ± 1.21	27.47 ± 1.51	943.44 ± 63.48	1.00×	-
	NVQLink (RDMA)	368 ± 15	0.96×	15.37 ± 0.30	25.83 ± 0.74	784.75 ± 48.96	1.20×	1.06×
	QCLINK (HTTP)	174 ± 14	2.03×	10.95 ± 0.29	21.35 ± 2.24	569.64 ± 15.83	1.66×	1.29×
	QCLINK (RDMA)	*142 ± 10*	*2.49×*	*3.43 ± 0.05*	*7.01 ± 1.05*	*177.40 ± 10.27*	*5.32×*	*3.92×*
GD	NVQLink (HTTP)	563 ± 49	1.00×	147.65 ± 1.82	188.81 ± 7.52	7677.73 ± 91.48	1.00×	-
	NVQLink (RDMA)	855 ± 66	0.66×	104.66 ± 0.63	149.12 ± 1.97	5114.33 ± 35.84	1.50×	1.26×
	QCLINK (HTTP)	184 ± 25	3.06×	31.32 ± 1.50	61.75 ± 13.59	1628.78 ± 79.38	4.71×	3.06×
	QCLINK (RDMA)	*184 ± 19*	*3.06×*	*18.88 ± 0.23*	*35.49 ± 9.61*	*982.65 ± 12.47*	*7.81×*	*5.32×*

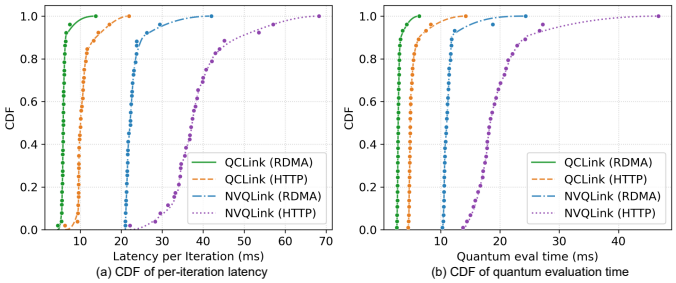

Fig. 5. CDF of iteration-level and evaluation-level latency across architectures.

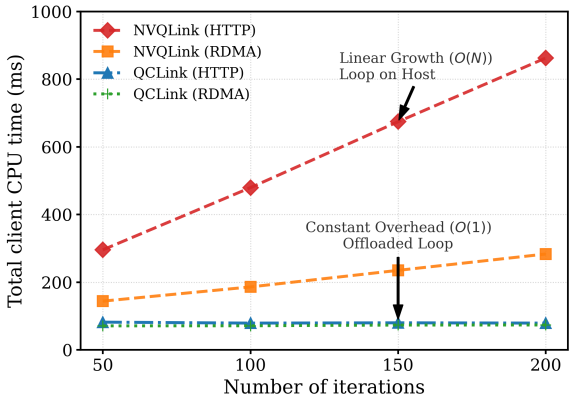

Fig. 6. Host CPU busy time as a function of iteration budget.

B. Overall Performance and Overhead Evaluation

We first evaluate QCLINK under single-job execution to quantify how offloading the optimization loop to the SmartNIC and accelerating the backend-facing path with RDMA affect latency, completion time, CPU overhead, and tail stability.

Per-Evaluation Latency. We begin with backend evaluation latency, since each optimization iteration repeatedly invokes one or more circuit evaluations and directly accumulates this cost. Figure 5 plots the CDF of per-evaluation latency across execution architectures. We observe that moving the backend-facing path from HTTP to RDMA shifts the distribution left and reduces variability, indicating that the SmartNIC-to-backend datapath becomes both faster and more predictable. This reduction is especially important for SPSA-like optimizers that issue a large number of short, latency-sensitive evaluations.

Per-Iteration Latency. We next evaluate per-iteration latency for SPSA-driven optimization across all baselines. Host-loop baselines expose the control loop to host OS scheduling and interrupt variability, while NIC-loop isolates the optimizer from host-side jitter. NIC-HTTP introduces an intermediate service layer but still relies on HTTP on the SmartNIC-backend path, limiting the reduction in per-evaluation overhead. In contrast, QCLINK reduces per-iteration latency by combining NIC-loop execution with RDMA-based evaluation dispatch. Table II summarizes the per-iteration latency across SPSA/COBYLA/GD and shows that QCLINK (NIC-RDMA) consistently achieves the lowest average latency. Speedup is computed against NVQLink (HTTP) under the same optimizer and workload configuration. Consistent with these results, Figure 5 shows that NIC-loop execution tightens the iteration-level latency distribution, while RDMA further reduces the long tail.

Classical Computation Overhead. In addition to backend evaluation and communication, iterative hybrid workloads incur optimizer-side control-plane cost per iteration, such as parameter update and state bookkeeping. Table II reports the average classical computation(CC) latency, showing that QCLINK consistently reduces CC overhead across optimizers. Compared to NVQLink (Host-HTTP), QCLINK achieves 1.84×-3.06× lower CC latency, indicating that SmartNIC-loop execution streamlines classical control logic and reduces host involvement. In contrast, replacing HTTP with RDMA alone does not necessarily reduce CC latency, since RDMA primarily accelerates the backend-facing datapath rather than the optimizer computation itself.

End-to-End Completion Time. We next examine end-to-end completion time, which captures the cumulative impact of per-iteration latency, synchronization overhead, and occasional slow iterations. Across tested optimizers, QCLINK significantly reduces completion time compared to host-centric baselines (Table II). While RDMA reduces per-evaluation overhead, the dominant gains come from eliminating host-side control-plane involvement by executing the optimization loop within the SmartNIC service layer, turning a fine-grained host-driven iteration process into a coarse-grained job submission model.

Host CPU Utilization. To quantify offloading benefits, we measure the host CPU busy time as the iteration budget increases. In host-centric baselines (NVQLink), the host remains on the per-iteration critical path, repeatedly orchestrating circuit evaluations, processing responses, and synchronizing optimizer progress. As a result, host-side CPU work accumulates across iterations. Figure 6 shows host CPU busy time as a function of the number of optimization iterations. We observe that NVQLink exhibits CPU busy time that grows approximately

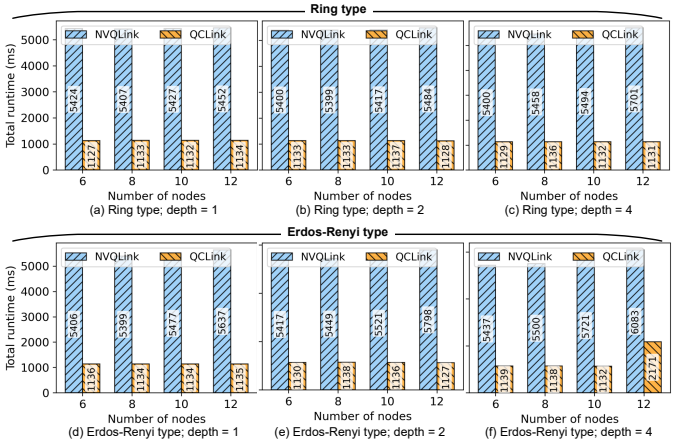

Fig. 7. End-to-end runtime comparison between NVQLink (host-loop) and QCLINK (SmartNIC-loop) across node sizes, graph families, and circuit depths.

linearly with the iteration count, indicating an $O(n)$ host-side overhead due to fine-grained control-loop involvement. In contrast, QCLINK offloads the optimization loop to the SmartNIC service layer, reducing host participation to coarse-grained job submission and result retrieval. Consequently, the host CPU busy time remains approximately constant (O(1)) as the iteration budget increases, demonstrating that QCLINK effectively eliminates per-iteration host overhead and enables the host to remain near-idle during optimization.

Tail Latency and Stability. Hybrid optimization loops are sensitive to tail latency, as occasional slow iterations can significantly delay convergence. Figure 5 shows that NIC-loop execution produces a tighter latency distribution than host-loop baselines, indicating reduced jitter and fewer straggling iterations. RDMA further reduces evaluation-level tail latency (lower subfigure), which directly translates into lower high-percentile per-iteration latency (Table II). Overall, QCLINK improves both average performance and execution stability under repeated circuit evaluations.

C. Sensitivity Evaluation

While prior experiments demonstrate that QCLINK reduces per-iteration latency and tail latency under fixed workloads, in practical deployments the optimization runtime is also affected by (i) problem scale (*e.g.,* number of nodes in MaxCut), (ii) circuit depth p, and (iii) non-trivial network RTT between the client and the execution service. To evaluate how these factors interact, we conduct a parameter sweep study under a fixed RTT of 20 ms and compare two execution architectures: NVQLink (host-loop) and QCLINK (SmartNIC-loop). We evaluate QAOA-MaxCut using SPSA with a fixed optimization budget of 50 iterations. We vary (i) the number of nodes $\{6, 8, 10, 12\}$, (ii) graph type $\{$*ring*, *Erdős-Rényi*$\}$, and (iii) circuit depth $p \in \{1, 2, 4\}$, resulting in 24 configurations per architecture. The network RTT between the client and the service layer is set to 20 ms. We report the end-to-end optimization runtime (`Total Runtime (ms)`) for both architectures, and additionally report the per-iteration delay distribution (`Iteration Duration (ms)`) for QCLINK.

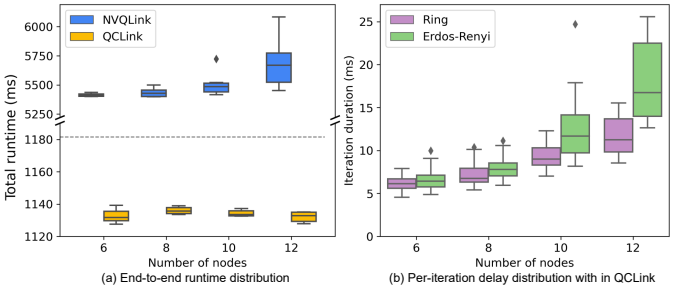

Fig. 8. Runtime distribution and per-iteration delay.

TABLE III
THROUGHPUT SCALABILITY UNDER CONCURRENCY ACROSS ALGORITHMS (JOBS/S)

Algorithm	*Architecture*	*C=1*	*C=2*	*C=4*	*C=8*	*C=16*
SPSA	NVQLink (HTTP)	0.08	0.21	0.44	0.71	1.05
	NVQLink (RDMA)	0.13	0.31	0.78	1.46	2.21
	QCLINK (HTTP)	*0.87*	1.22	1.44	1.77	2.66
	QCLINK (RDMA)	0.80	*1.83*	*2.47*	*3.52*	*3.96*
	Speedup	*10.00×*	*8.71×*	*5.61×*	*4.96×*	*3.77×*
COBYLA	NVQLink (HTTP)	0.16	0.18	0.19	0.19	0.19
	NVQLink (RDMA)	0.35	0.41	0.43	0.45	0.58
	QCLINK (HTTP)	0.86	1.22	1.76	1.99	2.25
	QCLINK (RDMA)	*1.80*	*3.59*	*3.60*	*4.68*	*5.52*
	Speedup	*11.25×*	*19.94×*	*18.95×*	*24.63×*	*29.05×*
GD	NVQLink (HTTP)	0.12	0.17	0.23	0.28	0.35
	NVQLink (RDMA)	0.18	0.27	0.30	0.39	0.50
	QCLINK (HTTP)	0.45	0.53	0.61	0.69	0.68
	QCLINK (RDMA)	*0.56*	*0.94*	*1.06*	*1.11*	*1.16*
	Speedup	*4.67×*	*5.53×*	*4.61×*	*3.96×*	*3.31×*

Host-Loop Execution with RTT. The key difference between the two architectures is the frequency at which the optimization control loop crosses the network boundary. In NVQLink (host-loop), the client executes the full SPSA loop and triggers two backend evaluations per iteration (corresponding to θ^+ and θ^-), introducing approximately $2 \times 50 = 100$ RTT-sensitive round trips in total. In contrast, QCLINK offloads the SPSA loop into the SmartNIC service layer: the client submits the job once and only performs coarse-grained interactions (submit, polling, and retrieving results), making the end-to-end runtime significantly less sensitive to RTT.

End-to-End Runtime Comparison across Scale and Depth. Figure 7 presents the end-to-end runtime under different MaxCut graph sizes (number of nodes), graph families, and QAOA circuit depths. Across all configurations, QCLINK consistently outperforms NVQLink, and the performance gap becomes more pronounced as circuit depth increases. This trend suggests that host-loop execution incurs increasing cumulative overhead from repeated RTT-sensitive synchronization and control-plane involvement, whereas SmartNIC-loop execution amortizes client-side communication by keeping the optimizer and evaluation dispatch local to the service layer.

Runtime Variability and Distribution. To further assess performance robustness, Figure 8(a) summarizes the distribution of end-to-end runtime across all configurations. QCLINK exhibits a lower median runtime and a tighter interquartile range compared to NVQLink, indicating improved stability when running a diverse mix of problem instances under non-trivial RTT. This result is consistent with our earlier observation

TABLE IV
GENERALITY BEYOND QAOA: PERFORMANCE SUMMARY ACROSS HYBRID AND EVALUATION-INTENSIVE WORKLOADS

Workload	*Per-iter (ms)*		*P99 (ms)*		*Completion Time (ms)*		*Improvement*	
	NVQLink	QCLINK	*NVQLink*	QCLINK	*NVQLink*	QCLINK	*CT Speedup*	*P99 Improv.*
VQE	40.42 ± 1.20	6.39 ± 0.45	61.12 ± 7.55	9.86 ± 2.37	1980.95 ± 72.63	419.02 ± 12.84	4.73 ×	6.19×
QNN	232.68 ± 9.54	105.77 ± 6.29	359.90 ± 41.87	197.32 ± 25.84	11056.50 ± 523.31	5250.06 ± 313.18	2.11×	1.82×
QSVM	212.47 ± 8.13	90.56 ± 5.92	333.08 ± 39.71	171.24 ± 26.45	10524.87 ± 497.60	4930.82 ± 284.39	2.13×	1.94×
QKNN	187.21 ± 7.95	88.64 ± 5.10	293.57 ± 30.09	156.28 ± 19.33	9255.61 ± 429.04	4401.18 ± 256.73	2.10×	1.88×
RB	124.29 ± 6.15	78.82 ± 3.96	197.54 ± 18.17	120.34 ± 17.91	6114.12 ± 301.56	3536.83 ± 243.27	1.73×	1.65×
XEB	9858.05 ± 215.48	9202.19 ± 189.84	15225.22 ± 982.65	14799.08 ± 910.77	49468.49 ± 948.13	45487.36 ± 893.62	1.09×	1.03×

that isolating the optimization loop from host OS scheduling reduces variance and mitigates tail amplification.

Per-Iteration Delay Characterization in SmartNIC-Loop. Figure 8(b) shows the per-iteration delay distribution within QCLINK grouped by graph type and number of graph nodes. We observe that iteration delays remain relatively stable across node sizes, while increasing depth p leads to higher iteration cost due to increased graph complexity and backend evaluation workload. Overall, these results indicate that QCLINK effectively decouples end-to-end convergence time from RTT-amplified host-side control overhead and exposes a more predictable iteration-level execution model.

D. Concurrent Workloads

We evaluate scalability by running multiple optimization jobs concurrently. As concurrency increases, host-centric baselines experience rapid degradation in latency and throughput due to CPU contention and scheduling overhead. Table III reports throughput (jobs/s) as we increase the number of concurrent optimization jobs from $C = 1$ to $C = 16$. Overall, QCLINK scales substantially better than host-centric baselines across all three optimizers (SPSA, COBYLA, and GD), because the SmartNIC service layer multiplexes multiple optimization loops without requiring per-iteration host-side orchestration. In contrast, host-loop baselines must repeatedly execute control logic, synchronize evaluation requests, and handle interrupts on the host CPU, which becomes a scalability bottleneck as concurrency increases. Across optimizers, QCLINK (NIC-RDMA) achieves the highest throughput at moderate-to-high concurrency. For example, at $C = 16$, QCLINK (NIC-RDMA) reaches 3.96 jobs/s (SPSA), 5.52 jobs/s (COBYLA), and 1.16 jobs/s (GD), consistently outperforming both NVQLink (HTTP) and NVQLink (RDMA). This demonstrates that eliminating fine-grained host involvement provides multiplicative gains under load, beyond the benefits of simply replacing HTTP with RDMA. We also observe that different optimizers exhibit different scaling behaviors. SPSA scales more smoothly because its iteration structure is highly regular and dominated by repeated evaluations, allowing the SmartNIC scheduler to efficiently pipeline requests across jobs. In contrast, COBYLA shows limited scalability in the host-loop baseline, suggesting that its control path is more synchronization-heavy and becomes RTT- and CPU-bound under concurrency. GD achieves lower absolute throughput because its per-iteration computation cost is higher, reducing the number of jobs that can be completed per unit time even with offloading. Comparing QCLINK (NIC-HTTP) and QCLINK (NIC-RDMA) in Table III, we find that RDMA consistently improves throughput as concurrency increases. This is because at high C, the SmartNIC must sustain a high request rate to the backend, and RDMA reduces per-evaluation CPU overhead and protocol processing on the backend-facing path, preventing the communication stack from becoming the limiting factor.

E. Extended Experiments

To assess generality, we evaluate QCLINK on additional workloads spanning variational training/optimization (VQE, QNN), kernel- and distance-based learning (QSVM, QKNN), and backend characterization (RB, XEB). For each workload, we compare NVQLink (HTTP) against QCLINK (NIC-RDMA) and report per-iteration latency, p99 latency, and end-to-end completion time. We keep hardware, backend, and network settings consistent with §V-B unless stated otherwise, and use the same default shot configuration as in our QAOA experiments. We define an *iteration* as one logical synchronization step: an optimizer step for VQE/QNN, one kernel (or distance) evaluation round for QSVM/QKNN, and one measurement round for RB/XEB. Completion time is the wall-clock time to complete a fixed workload-specific budget. Unless noted, we use 50 iterations/rounds; for XEB we use 5 rounds due to its sampling-dominated cost (10^6 shots per instance). QNN/QSVM/QKNN use evaluation-intensive batches (four batches per iteration, 496 circuits total); RB dispatches 200 random circuits per round; XEB dispatches 32 instances per round. Table IV shows that QCLINK improves performance across all workloads: 1.07×-6.33× lower per-iteration latency, 1.03×-6.19× better p99 latency, and 1.09×-4.73× lower completion time relative to NVQLink. The largest gains arise in tightly-coupled hybrid loops with frequent backend interactions (*e.g.,* VQE: 6.33× lower iteration latency and 4.73× lower completion time), where host-side control and per-request overheads are a substantial fraction of the critical path. Batch-style learning workloads (QNN/QSVM/QKNN) see moderate but consistent improvements because circuit execution dominates total time, yet QCLINK reduces per-batch orchestration and tail effects. Characterization workloads benefit based on their overhead share: RB improves noticeably due to repeated dispatch or collection, while XEB improves modestly since runtime is dominated by backend sampling.

VI. DISCUSSION

Applicability and Iteration Granularity. We have evaluated QCLINK across a diverse set of hybrid and evaluation-intensive workloads, including variational optimization, quantum learning, and backend characterization [4], [17]. Our results suggest that QCLINK is most effective when a workload exposes a

fine-grained, latency-sensitive iteration structure, where per-iteration control overhead and its tail variability are repeatedly amplified. In contrast, workloads with *coarse-grained* iterations (i.e., fewer synchronization points and heavier work per iteration [17], [54]) naturally amortize control overhead, leading to smaller offloading benefits. A promising direction is to further restructure such workloads by splitting monolithic iterations into more frequent synchronization steps (*e.g.,* finer batching and incremental updates) [26], [14], [53], which increases control-loop repetition and better matches QCLINK's in-network execution model.

Relation to Tight Coupling. Recent systems pursue tighter coupling between classical accelerators and quantum hardware to support microsecond-scale feedback paths (*e.g.,* error correction and real-time control) [10], [11], [46], [43]. Our work targets a complementary layer: distributed hybrid optimization loops that run at a coarser granularity but suffer from frequent, fine-grained control interactions and repeated host mediation [19], [5]. By relocating the optimization loop to the network data plane, QCLINK reduces control-plane overheads and tail-latency amplification in iterative training [54], [37], [33]. These directions are orthogonal and could be combined to simultaneously support real-time quantum control and efficient distributed hybrid optimization [41], [43].

Trade-Offs and Outlook. Data-plane offloading introduces trade-offs. SmartNIC computation and memory are limited, motivating our focus on low arithmetic intensity and small state per-job; richer optimizers may need hybrid execution [26], [14]. SmartNIC-resident services also raise engineering complexity (programming, debugging, and verification), increasing the importance of clean abstractions and robust control-plane interfaces [27], [57]. Finally, multi-tenant deployments require scheduling and isolation on the SmartNIC to avoid contention and preserve fairness [14], [60], [26]. Our current prototype assumes RDMA-accessible backends and focuses on single-backend execution; it does not address dynamic backend discovery or heterogeneous networks [54], [25]. Future work includes multi-backend scheduling [45], adaptive batching of quantum evaluations, and integration with accelerator-centric runtimes to coordinate GPUs, SmartNICs, and quantum hardware [41], [52].

VII. Related Work

Hybrid Quantum-Classical Algorithms and Systems. Hybrid quantum-classical algorithms (*e.g.,* variational algorithms and gradient-free optimizers such as SPSA) are a practical paradigm for near-term devices [38], [34], [7], [39], [4]. Existing work largely focuses on algorithm design, noise robustness, and convergence [39], [20], alongside software frameworks that orchestrate classical-quantum interactions [1], [3], [58]. Most frameworks follow a host-centric model: the optimizer runs on CPUs/accelerators and invokes quantum evaluations via RPCs, task schedulers, or service APIs [30], [11]. While flexible and portable, this design leaves iterative workloads exposed to repeated control-plane interactions and host-side overheads [19], [33]. Our work complements these efforts by reconsidering *where* the optimization loop should execute and by reducing distributed control overhead via offloading.

Tight Coupling and Quantum Control Accelerator. Systems for quantum error correction and real-time feedback have pursued tight coupling between classical accelerators and quantum hardware, using high-speed interconnects and accelerator-aware runtimes to minimize measurement-to-actuation latency [10], [11], [43], [41], [46]. These designs primarily target data-intensive, microsecond-scale feedback paths and often assume powerful accelerators (*e.g.,* GPUs). In contrast, we target hybrid optimization loops at a higher abstraction level, where the dominant cost is frequently distributed control-plane overhead rather than raw computation. Offloading lightweight optimizer logic to SmartNICs addresses this complementary bottleneck and can coexist with accelerator-centric control mechanisms.

Quantum Cloud and Distributed Quantum Computing. Cloud quantum platforms motivate research on remote execution, resource sharing, and multi-tenant scheduling, with emphasis on throughput, fairness, and usability [12], [42], [47], [23], [24], [49], [50], [51]. Our work is aligned with this setting but focuses on a different inefficiency: in iterative hybrid workloads, fine-grained distributed control loops can dominate end-to-end latency and amplify tail effects [5]. We show that executing the optimization loop as an in-network service provides a new mechanism to improve efficiency within distributed quantum infrastructures.

In-Network Computing and SmartNIC Offloading. Programmable switches and SmartNICs have been used to offload host logic for aggregation, caching, synchronization, and protocol processing in latency-sensitive distributed systems [22], [16], [17], [27], [2], [15], [28], [55], [59], [56]. We build on this line of work by applying data-plane offloading to hybrid quantum-classical optimization. Different from many stateless or stream-oriented in-network tasks, our design executes *stateful* iterative control logic on the SmartNIC, demonstrating that programmable network devices can host latency-critical control-plane functions for emerging heterogeneous workloads [26], [14], [45], [54].

VIII. Conclusion

This paper presents a SmartNIC-based framework that offloads hybrid optimization loops to the network data plane and leverages RDMA for low-latency, CPU-bypassed communication with quantum backends. By executing parameter updates and iteration control on the SmartNIC, our design removes repeated host mediation from the critical path, reducing per-iteration latency, improving completion time, and scaling better under concurrent workloads.

Acknowledgments

This research was supported by Ningbo Yongjiang Talent Introduction Programme.

REFERENCES

[1] G. Aleksandrowicz et al. Qiskit: An open-source framework for quantum computing. Zenodo (Software), Version 0.7.2, 2019.

[2] M. Blöcher et al. Switches for hire: Resource scheduling for data center in-network computing. In *ASPLOS*, pages 268–285, 2021.

[3] M. Broughton et al. Tensorflow quantum: A software framework for quantum machine learning. arXiv:2003.02989, 2020.

[4] M. Cerezo et al. Variational quantum algorithms. *Nat. Rev. Phys.*, 3(9):625–644, 2021.

[5] J. Dean and L. A. Barroso. The tail at scale. *Commun. ACM*, 56(2):74–80, 2013.

[6] A. Dragojević et al. FaRM: Fast remote memory. In *NSDI*, pages 401–414, 2014.

[7] E. Farhi, J. Goldstone, and S. Gutmann. A quantum approximate optimization algorithm. arXiv:1411.4028, 2014.

[8] R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[9] R. T. Fielding, M. Nottingham, and J. Reschke. HTTP Semantics. RFC 9110, June 2022.

[10] X. Fu et al. An experimental microarchitecture for a superconducting quantum processor. In *MICRO*, pages 813–825, 2017.

[11] X. Fu et al. eqasm: An executable quantum instruction set architecture. In *HPCA*, pages 224–237, 2019.

[12] E. Giortamis et al. QOS: Quantum operating system. In *OSDI*, 2025.

[13] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42(6):1115–1145, 1995.

[14] S. Grant et al. Smartnic performance isolation with fairnic: Programmable networking for the cloud. In *SIGCOMM*, pages 681–693, 2020.

[15] Y. He et al. A generic service to provide in-network aggregation for key-value streams. In *ASPLOS*, pages 33–47, 2023.

[16] X. Jin et al. Netcache: Balancing key-value stores with fast in-network caching. In *SOSP*, pages 121–136, 2017.

[17] X. Jin et al. Netchain: Scale-free sub-RTT coordination. In *NSDI*, pages 35–49, 2018.

[18] K. Kaffes et al. Shinjuku: Preemptive scheduling for μsecond-scale tail latency. In *NSDI*, pages 345–360, 2019.

[19] A. Kalia, M. Kaminsky, and D. G. Andersen. Datacenter RPCs can be general and fast. In *NSDI*, pages 1–16, 2019.

[20] A. Kandala et al. Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets. *Nature*, 549(7671):242–246, 2017.

[21] B. Li et al. Clicknp: Highly flexible and high-performance network processing with reconfigurable hardware. In *SIGCOMM*, pages 1–14, 2016.

[22] B. Li et al. KV-Direct: High-performance in-memory key-value store with programmable NIC. In *SOSP*, pages 137–152, 2017.

[23] T. Li and Z. Zhao. Moirai: Optimizing quantum serverless function orchestration via device allocation and circuit deployment. In *IEEE ICWS*, pages 707–717, 2024.

[24] T. Li, Z. Zhao, and J. Yin. Empowering quantum serverless circuit deployment optimization via graph contrastive learning and learning-to-rank co-designed approaches. In *IJCAI*, pages 9250–9258, 2025.

[25] T. Li, Z. Zhao, and J. Yin. Fortuna: Towards efficient selection of high-fidelity link for quantum network in the wild. In *INFOCOM*, pages 1–10. IEEE, 2025.

[26] J. Lin et al. Panic: A high-performance programmable nic for multi-tenant networks. In *OSDI*, pages 243–259, 2020.

[27] J. Liu et al. p4v: practical verification for programmable data planes. In *SIGCOMM*, pages 490–503, 2018.

[28] S. Liu et al. In-network aggregation with transport transparency for distributed training. In *ASPLOS*, pages 376–391, 2023.

[29] Z. Liu, Z. Zhao, and F. Zhang. Regimeguard: Continual learning queue scheduling for socially critical web services. In *WWW*, pages 9779–9787. ACM, 2026.

[30] P. Murali et al. Full-stack, real-system quantum computer studies: Architectural comparisons and design insights. In *ISCA*, pages 527–540, 2019.

[31] J. Nocedal and S. J. Wright. *Numerical Optimization*. New York, NY, USA, 2 edition, 2006.

[32] NVIDIA. NVIDIA NVQLink Architecture Integrates Accelerated Computing with Quantum Processors. NVIDIA Dev. Blog, 2025. Accessed: 2026-01-22.

[33] A. Ousterhout et al. Shenango: Achieving high CPU efficiency for latency-sensitive datacenter workloads. In *NSDI*, pages 361–378, 2019.

[34] A. Peruzzo et al. A variational eigenvalue solver on a photonic quantum processor. *Nat. Commun.*, 5:4213, 2014.

[35] S. Peter et al. Arrakis: The operating system is the control plane. In *OSDI*, pages 1–16, Broomfield, CO, USA, 2014.

[36] M. J. D. Powell. A direct search optimization method that models the objective and constraint functions by linear interpolation. In S. Gomez and J.-P. Hennart, editors, *Adv. Optim. Numer. Anal.*, volume 275 of *Math. Appl.*, pages 51–67. Dordrecht, 1994.

[37] G. Prekas, M. Kogias, and E. Bugnion. Zygos: Achieving low tail latency for microsecond-scale networked tasks. In *SOSP*, pages 325–341, 2017.

[38] J. Preskill. Quantum computing in the nisq era and beyond. *Quantum*, 2:79, 2018.

[39] J. C. Spall. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE Trans. Autom. Control*, 37(3):332–341, 1992.

[40] W. Tang et al. Cutqc: Using small quantum computers for large quantum circuit evaluations. In *ASPLOS*, pages 473–486, 2021.

[41] C. Tao et al. Qtenon: Towards low-latency architecture integration for accelerating hybrid quantum-classical computing. In *ISCA*, pages 299–312, 2025.

[42] R. Tao et al. Quantum virtual machines. In *OSDI*, 2025.

[43] W. Tian et al. Artery: Fast quantum feedback using branch prediction. In *ISCA*, pages 285–298, 2025.

[44] S. Tsai and Y. Zhang. LITE kernel RDMA support for datacenter applications. In *SOSP*, pages 306–324. ACM, 2017.

[45] S. S. Udayashankar et al. Draconis: Network-accelerated scheduling for microsecond-scale workloads. In *EuroSys*, 2024.

[46] S. Vittal, P. Das, and M. K. Qureshi. Astrea: Accurate quantum error-decoding via practical minimum-weight perfect-matching. In *ISCA*, pages 2:1–2:16, 2023.

[47] M. Wang, P. Das, and P. J. Nair. Qoncord: A multi-device job scheduling framework for variational quantum algorithms. In *MICRO*, pages 735–749, 2024.

[48] A. Xilinx. Alveo u280 data center accelerator card data sheet (ds963). *Last accessed*, pages 06–01, 2023.

[49] X. Yue et al. Demeter: Fine-grained function orchestration for geo-distributed serverless analytics. In *IEEE INFOCOM*, pages 2498–2507. IEEE, 2024.

[50] X. Yue et al. Hyfaas: Accelerating serverless workflows by unleashing hybrid resource elasticity. *IEEE Transactions on Parallel and Distributed Systems*, 37(1):272–286, 2025.

[51] X. Yue, S. Yang, F. Li, Y. Li, and Y. Wang. Rocket: Warming serverless inference via hierarchical ML artifact pre-loading and sharing. In *IEEE Conference on Computer Communications*, 2026.

[52] X. Yue, F. Zhao, F. Ye, J. Yu, Z. Li, T. Li, Z. Zhao, and J. Yin. Heterosim: Towards high-fidelity heterogeneous llm training simulation on gpus. In *Proceedings of the ACM Web Conference 2026*, pages 5189–5197, 2026.

[53] X. Yue, Z. Zhao, J. Yu, H. Ma, Z. Li, T. Li, and J. Yin. sMoE: Elastic MoE-Based Inference with Serverless Computing via Fine-Grained Expert Scaling. In *ACM/IEEE DAC*, 2026.

[54] J. Zhang et al. Rpcnic: Enabling efficient datacenter rpc offloading on pcie-attached smartnics. In *HPCA*, pages 1379–1394, 2025.

[55] Z. Zhao et al. Effective ddos mitigation via ml-driven in-network traffic shaping. *IEEE Transactions on Dependable and Secure Computing*, 21(4):4271–4289, 2024.

[56] Z. Zhao et al. Verify all traffic: Towards zero-trust in-network intrusion detection against multipath routing. *IEEE Journal on Selected Areas in Communications*, 43(6):2155–2171, 2025.

[57] Z. Zhao, T. Li, Z. Li, and J. Yin. Relational verification for cost-aware quantum program optimization. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(17):14414–14422, 2026.

[58] Z. Zhao, T. Li, and J. Yin. Quamap: A multi-backend benchmark dataset for quantum circuit mapping and learning-based compiler evaluation. In *KDD (1)*, pages 2913–2923. ACM, 2026.

[59] Z. Zhao, Z. Li, Z. Song, F. Zhang, and B. Chen. RIDS: Towards advanced IDS via RNN model and programmable switches co-designed approaches. In *IEEE INFOCOM 2024 - IEEE Conference on Computer Communications*, pages 591–600, 2024.

[60] Y. Zhou et al. Smartnic security isolation in the cloud with s-nic. In *EuroSys*, pages 851–869, 2024.