

Politecnico di Torino

ScuDo
Scuola di Dottorato ~ Doctoral School
WHAT YOU ARE, TAKES YOU FAR

Doctoral Dissertation
Doctoral Program in Control and Computer Engineering (38^{th}cycle)

Full-Stack Implementation and Platform-Aware Optimization of Hybrid Quantum–Classical Methods on NISQ Devices

Applications in Quantum Machine Learning and Combinatorial Optimization

By

Giacomo Vitali

******

Supervisor(s):
Prof. Bartolomeo Montrucchio, Supervisor
Dr. Olivier Terzo, Co-Supervisor

Politecnico di Torino
2026

Declaration

I hereby declare that, the contents and organization of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

Giacomo Vitali
2026

* This dissertation is presented in partial fulfillment of the requirements for **Ph.D. degree** in the Graduate School of Politecnico di Torino (ScuDo).

For my wife Elena, the reason why I get up every morning, and my daughter Amanda, the real and only masterpiece of my life.

Acknowledgements

This dissertation would not have been possible without the support, expertise, and generosity of many individuals and institutions.

The experimental work reported here relied on access to quantum hardware provided by QuEra Computing, Pasqal, and IQM Quantum Computers. I am grateful to these companies not only for granting machine access but also for the technical discussions that informed the design and interpretation of the experiments.

Access to quantum and classical computing resources was made possible through research credits provided by the Pawsey Supercomputing Centre, CINECA, and Amazon Web Services. These allocations were essential for carrying out the large-scale emulation and hardware experiments described in this thesis.

The work on resource management for hybrid quantum–HPC systems was carried out within the *SmartHPCQC* project, funded by the European Union NextGenerationEU programme through the ICSC National Centre for HPC, Big Data and Quantum Computing (CN00000013, MUR Act n. 1031, 17/06/2022). I thank all project partners, namely Politecnico di Torino, LINKS Foundation, Università di Torino, E4 Computing Engineering, CINECA, and the Barcelona Supercomputing Centre (BSC), for their contributions to the research on resource allocation and hybrid execution strategies.

The Quantum Competence Center of Intesa Sanpaolo provided valuable collaboration and support throughout the Quantum Reservoir Computing case study. Their engagement from an applied financial perspective shaped the framing of the credit-card default prediction task and enriched the experimental evaluation.

Finally, I am grateful to my colleagues at LINKS Foundation and Politecnico di Torino, in particular to my supervisors Prof. Montrucchio and Dr. Terzo, and to my peer Chiara Vercellino for their daily support, technical exchanges, and

encouragement throughout the course of this doctoral research. Their contributions, both direct and indirect, are reflected throughout this dissertation.

Abstract

Current quantum processors, Noisy Intermediate-Scale Quantum (NISQ) devices, offer tens to a few hundred qubits with limited coherence and non-negligible error rates. In this regime, purely quantum solutions are not feasible for most practical problems. Hybrid quantum–classical algorithms, where a quantum processor executes specific computational kernels within a classical workflow, have emerged as the most promising near-term approach. However, the performance of such hybrid methods depends not only on circuit design but on the entire computational stack: execution models, resource management policies, and platform-level integration within High-Performance Computing (HPC) infrastructures.

This dissertation investigates hybrid quantum–classical computation from a cross-layer perspective, spanning algorithmic design, workload characterization, resource management, and platform integration. The central argument is that meaningful quantum-enhanced computation in the NISQ regime does not arise from isolated algorithmic improvements. It emerges from coordinated co-design across abstraction layers: algorithms matched to hardware capabilities, execution models that account for hybrid workflow latency and asymmetry, and resource management strategies reflecting the operational realities of shared, heterogeneous HPC–QC systems.

Three case studies are developed and evaluated. The first applies Quantum Reservoir Computing (QRC) to credit-card default prediction, using a 256-qubit neutral-atom platform as a nonlinear feature transformer within a classical classification pipeline. The second introduces the Quantum Evolution Kernel (QEK), a hardware-native graph kernel for protein graph classification that exploits analog quantum dynamics to construct similarity measures between graphs. The third presents BBQ-mIS, a hybrid Branch-and-Bound solver for graph coloring that uses a neutral-atom device as a maximal Independent Set generator within a classical decomposition framework.

Three complementary resource management strategies are introduced and evaluated. The vQPU (Virtual Quantum Processing Unit) abstraction enables time-based multiplexing of a physical QPU across concurrent hybrid jobs within a SLURM-compatible HPC environment, demonstrated on a real gate-based device at Politecnico di Torino integrated with the Leonardo supercomputer at CINECA. Workflow decomposition reduces idle classical reservations by separating quantum and classical phases into independently scheduled steps. Dynamic malleability allows running applications to release and reacquire classical nodes in response to changing phase requirements, improving cluster utilization without requiring explicit workflow declarations.

Experimental evaluation spans noiseless statevector emulation, shot-based simulation, and real-device execution on QuEra's Aquila neutral-atom platform. QRC achieves classification scores competitive with a deep neural network baseline in emulation, with hardware degradation absorbed by the classical readout at larger data scales. QEK matches the Shortest Path Kernel on the PROTEINS benchmark (65.6% vs. 65.3% F1) under both emulation and hardware conditions. BBQ-mIS reproduces optimal graph colorings across all emulation instances; real-device execution on Aquila yields non-trivial maximal Independent Set recovery rates, validating the neutral-atom device as a viable offline candidate generator within a classical solver.

The results do not support broad claims of quantum advantage. They demonstrate narrowly scoped usefulness under realistic NISQ conditions: the hardware access model and classical scaffolding are decisive factors in all three case studies. The cross-layer perspective developed in this dissertation is not a temporary workaround for immature hardware. It is a durable framework for reasoning about quantum computing as an integral component of future HPC infrastructures.

List of Publications

The following publications, produced during the course of this doctoral research, underpin the contributions presented in this dissertation.

Journal articles and preprints

G. Vitali, C. Vercellino, P. Viviani, O. Terzo, B. Montrucchio, V. Zaffaroni, F. Cibrario, C. Mattia, G. Ranieri, A. Sabatino, F. Bonazzi, and D. Corbelletto, "Quantum Reservoir Computing for Credit Card Default Prediction on a Neutral Atom Platform," *arXiv preprint arXiv:2510.04747*, 2025 [1]. Under review for publication in a peer-reviewed journal.

C. Vercellino, M. Y. Naghmouchi, W. Coelho, G. Vitali, A. Scionti, P. Viviani, O. Terzo, and B. Montrucchio, "Hybrid Quantum-Classical Branch-and-Price Method for the Vertex Coloring Problem," *arXiv preprint arXiv:2508.18887*, 2025 [2]. Under review for publication in a peer-reviewed journal.

M. Cipollini, S. Rizzo, S. Iserte, P. Viviani, G. Vitali *et al.*, "Three ways to share a QPU: Scheduling strategies for hybrid Quantum-HPC applications," *Future Generation Computer Systems*, vol. 185, p. 108699, 2026 [3].

P. Viviani, F. Bertone, G. Vitali *et al.*, "Lagrange: Operating Italy's First Publicly-Accessible Quantum Computer for Research and Education," *arXiv preprint arXiv:2604.21695*, 2026 [4]. Under review at a peer-reviewed conference venue.

Conference papers

E. Giusto, G. Iurlaro, B. Montrucchio, A. Scionti, O. Terzo, C. Vercellino, G. Vitali, and P. Viviani, "Harnessing a 256-qubit Neutral Atom Simulator for Graph Classification," in *Proc. IEEE International Conference on Quantum Computing and Engineering (QCE)*, vol. 01, pp. 296–305, 2024 [5].

C. Vercellino, G. Vitali, P. Viviani, E. Giusto, A. Scionti, A. Scarabosio, O. Terzo, and B. Montrucchio, "BBQ-mIS: A Parallel Quantum Algorithm for Graph Coloring Problems," in *Proc. IEEE International Conference on Quantum Computing and Engineering (QCE)*, vol. 02, pp. 141–147, 2023 [6].

P. Viviani, R. Rocco, M. Barbieri *et al.*, "Assessing the Elephant in the Room in Scheduling for Current Hybrid HPC-QC Clusters," in *Proc. 55th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pp. 184–187, 2025 [7].

R. Rocco, S. Rizzo, M. Barbieri *et al.*, "Dynamic Solutions for Hybrid Quantum-HPC Resource Allocation," in *Proc. IEEE International Conference on Quantum Computing and Engineering (QCE)*, vol. 02, pp. 34–40, 2025 [8].

C. Vercellino, P. Viviani, G. Vitali, A. Scionti, A. Scarabosio, O. Terzo, E. Giusto, and B. Montrucchio, "Neural-powered Unit Disk Graph Embedding: Qubits Connectivity for Some QUBO Problems," in *Proc. IEEE International Conference on Quantum Computing and Engineering (QCE)*, pp. 186–196, 2022 [9].

C. Vercellino, G. Vitali, P. Viviani, A. Scionti, A. Scarabosio, O. Terzo, E. Giusto, and B. Montrucchio, "Neural Optimization for Quantum Architectures: Graph Embedding Problems with Distance Encoder Networks," in *Proc. IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC)*, pp. 380–389, 2023 [10].

C. Vercellino, G. Vitali, P. Viviani, A. Scionti, O. Terzo, and B. Montrucchio, "Harnessing DEN Models for Quantum Computing Tasks on Neutral Atom QPUs," in *Proc. IEEE International Conference on Quantum Computing and Engineering (QCE)*, vol. 02, pp. 217–222, 2024 [11].

G. Barillaro, A. Boella, F. Gandino, M. G. Vakili, E. Giusto, G. Mondo, B. Montrucchio, A. Scarabosio, A. Scionti, O. Terzo, and G. Vitali, "Com-

parison of Heuristic Approaches to PCI Planning for Quantum Computers," in *Proc. IEEE International Conference on Consumer Electronics (ICCE)*, pp. 1–6, 2023 [12].

One conference paper currently under submission, with the following preliminary title: "Digital Quantum Reservoir Computing for ATM Timeseries Prediction".

Other publications

Publications not directly related to the contributions of this dissertation:

P. Savio, A. Scionti, G. Vitali, P. Viviani, C. Vercellino, O. Terzo, H.-N. Nguyen, D. Magarielli, E. Spano, M. Marconcini, and F. Poli, "Accelerating Legacy Applications with Spatial Computing Devices," *The Journal of Supercomputing*, vol. 79, no. 7, pp. 7461–7483, 2023 [13].

C. Vercellino, A. Scionti, G. Varavallo, P. Viviani, G. Vitali, and O. Terzo, "A Machine Learning Approach for an HPC Use Case: the Jobs Queuing Time Prediction," *Future Generation Computer Systems*, vol. 143, pp. 215–230, 2023 [14].

F. Lubrano, C. Vercellino, G. Vitali, P. Viviani, A. Scionti, and O. Terzo, "Advanced Resource Allocation in the Context of Heterogeneous Workflows Management," in *Proc. 2nd Workshop on Workflows in Distributed Environments (WiDE '24)*, pp. 14–20, 2024 [15].

C. Vercellino, G. Vitali, P. Viviani, A. Scionti, O. Terzo, B. Montrucchio, P. J. Elahi, and U. Varetto, "Quantum-Assisted Design of Space-Terrestrial Integrated Networks," *arXiv preprint arXiv:2602.04350*, 2026 [16]. Under review for publication in a peer-reviewed journal.

Contents

List of Publications viii

List of Figures xix

List of Tables xxvi

List of Abbreviations xxx

1 Introduction 1

1.1 Hybrid Quantum–Classical Computing in the NISQ Era 3

1.2 Full-Stack Perspective: From Algorithms to Platforms 4

1.3 Research Questions and Thesis Contributions 6

1.4 Structure of the Thesis 8

2 Background and State of the Art 10

2.1 NISQ Devices and Hardware Constraints 10

2.1.1 Hardware Modalities and Architectural Trade-Offs 11

2.1.2 Noise Channels and Decoherence Mechanisms 12

2.1.3 Error Mitigation Techniques 14

2.1.4 Connectivity and Compilation Overheads 15

2.1.5 Toward Early Fault-Tolerant Systems 16

2.2 Variational and Hybrid Quantum Algorithms 18

2.2.1 General Structure 18
2.2.2 Shot Complexity and Resource Scaling 18
2.2.3 Distributed and Parallel VQA Strategies 19
2.3 Quantum Machine Learning in the NISQ Era 19
2.3.1 Data Encoding and Feature Maps 19
2.3.2 Trainability and Generalization 20
2.3.3 Dequantization and the Limits of Quantum Kernel Advantage 21
2.3.4 Quantum Reservoir Computing 22
2.3.5 Scalability Challenges 22
2.4 Quantum Algorithms for Combinatorial Optimization 23
2.4.1 QUBO and Ising Formulations 23
2.4.2 Quantum Approximate Optimization Algorithm 23
2.4.3 Quantum Annealing and Hybrid Solvers 24
2.4.4 Decomposition and Hybrid Cooperation 24
2.4.5 Complexity and Practical Limitations 24
2.4.6 Comparative Analysis of QML and Optimization Workloads 25
2.5 High-Performance Computing Systems and Batch Scheduling . . . 26
2.5.1 Batch Scheduling and Malleable Workloads 26
2.5.2 Heterogeneous Accelerator Integration: GPU and FPGA Co-Scheduling 26
2.5.3 The Quantum Integration Challenge: Novel Scheduling Constraints 27
2.6 Challenges in Quantum-HPC Integration 29
2.6.1 Latency Amplification 29
2.6.2 Resource Fragmentation and Idle Time 29
2.6.3 Co-Scheduling and Reservation Models 29
2.6.4 Virtualization and Abstraction Layers 30

2.7 Critical Assessment and Research Gaps 30
2.7.1 Algorithm-Hardware Mismatch 30
2.7.2 Underexplored Execution Models 30
2.7.3 Scalability Bottlenecks Beyond Qubits 31
2.7.4 Lack of Integrated QC–HPC Abstractions 31
2.7.5 Research Questions Emerging from the Gap Analysis 31

3 Quantum Machine Learning on NISQ Devices 33
3.1 Quantum Machine Learning in the NISQ Regime 33
3.1.1 Conceptual Foundations 34
3.1.2 Data Encoding Strategies 35
3.1.3 Trainability and Barren Plateaus 35
3.1.4 Computational Model of Hybrid QML 36
3.2 Variational Quantum Models and Quantum Neural Networks 37
3.2.1 Parameterized Quantum Circuits 37
3.2.2 Quantum Neural Networks 38
3.2.3 Training Strategies 38
3.2.4 Expressivity vs. Hardware Constraints 39
3.3 Quantum Reservoir Computing 39
3.3.1 From Classical to Quantum Reservoirs 40
3.3.2 Quantum Dynamical Model 40
3.3.3 Large-Scale Quantum Reservoir Learning with an Analog Quantum Computer 41
3.3.4 Case Study I: Classification with QRC 43
3.4 Quantum Kernel Methods 50
3.4.1 Implicit Feature Maps in Hilbert Space 51
3.4.2 Overlap Estimation Techniques 51

3.4.3 Quantum Evolution Kernel for Graph Learning 51
3.4.4 Case Study II: Graph Classification via Quantum Kernels . . 54
3.5 Comparative Analysis of QML Paradigms 57
3.6 Infrastructure-Level Implications of QML Workloads 60

4 Hybrid Quantum Approaches to Combinatorial Optimization 65
4.1 Combinatorial Optimization in the NISQ Era 65
4.2 QUBO and Ising Formulations 66
4.3 Variational Quantum Optimization 67
4.3.1 Hardware Constraints and Embedding 68
4.4 Hybrid Decomposition and Classical-Quantum Cooperation 68
4.4.1 Parallelization and Resource Management 69
4.4.2 Error Mitigation and Performance Metrics 69
4.5 Case Study III: BBQ-mIS: Solving the Graph Coloring Problem . . 70
4.5.1 Problem Formulation 70
4.5.2 Algorithmic Design of BBQ-mIS 71
4.5.3 Quantum MIS Subroutine 72
4.5.4 Branch-and-Bound Formulation 73
4.5.5 Parallelization and HPC Considerations 74
4.5.6 Methodological Relevance within the Thesis 74
4.6 Comparative Analysis of Hybrid Optimization Paradigms 76
4.7 Infrastructure-Level Implications of Optimization Workloads 79

5 Efficient Resource Management for Hybrid QC–HPC Systems 85
5.1 Hybrid Execution in the NISQ Era 85
5.2 Architectural Requirements for Efficient Hybrid Execution 88
5.2.1 R1: Compatibility with operational HPC control planes . . . 89

5.2.2 R2: Support for multiple integration modes 90
5.2.3 R3: Mechanisms to mitigate workload imbalance 90
5.2.4 R4: Separation of concerns between application, runtime, and scheduler 91
5.2.5 R5: Observability, predictability, and fairness under contention 91
5.3 Quantum Resource Virtualization: The vQPU Approach 92
5.4 Performance Characterization of vQPU 94
5.4.1 Experimental setup and workload model 94
5.4.2 Metrics 97
5.4.3 Observed trends and interpretation 97
5.5 Malleability and Workflows for Hybrid QC–HPC Applications . . . 100
5.5.1 Workflow-based resource allocation 101
5.5.2 Malleability-based resource allocation 102
5.5.3 Use-case driver: clustering aggregation as a hybrid QC–HPC workload 104
5.6 Performance Characterization of Dynamic Solutions 105
5.6.1 Experimental setup and execution scenarios 106
5.6.2 Metrics 107
5.6.3 Workload model and emulated quantum time scales 108
5.6.4 Observed trends and results 108
5.7 Comparative Analysis of Execution Strategies 111

6 Experimental Evaluation 116

6.1 Experimental Platforms and Testbeds 117
6.1.1 General execution model 117
6.1.2 Platforms for the QRC case study on credit-card default prediction 118
6.1.3 Platforms for the QEK case study on graph classification . . 119

6.1.4 Platforms for the BBQ-mIS case study on graph coloring . . 120
6.1.5 A note on remote access and practical reproducibility 121
6.2 Detailed Methodology 121
6.2.1 Common evaluation methodology 122
6.2.2 Methodology for the QRC credit-default study 122
6.2.3 Methodology for the QEK graph-classification study 130
6.2.4 Methodology for the BBQ-mIS graph-coloring study 137
6.3 Evaluation of QML Use-Cases 143
6.3.1 Results for the classification with QRC case study 144
6.3.2 Results for the QEK graph-classification case study 148
6.3.3 Comparative interpretation of the two QML case studies . . 152
6.4 Evaluation of Optimization Use-Cases 153
6.4.1 Interpretation of the optimization results 159
6.5 Performance, Scalability, and Resource Utilization 160
6.6 Discussion of Results 164

7 Discussion **169**
7.1 Lessons from QML and Optimization Use-Cases 170
7.2 Lessons from Cross-Layer Integration 172
7.3 Toward Production-Grade Hybrid QC–HPC Systems 174
7.4 Outlook Beyond the NISQ Era 176

8 Conclusions **179**
8.1 Answers to the Research Questions 179
8.2 Summary of Contributions 181
8.3 Limitations 183
8.4 Future Work 184

References **187**

Appendix A Details of Implemented Software Components **201**

A.1 DEN Model for Unit-Disk Graph Embedding 201

A.1.1 The Constrained Unit Disk Graph Problem 201

A.1.2 DEN Architecture and Training Procedure 202

A.1.3 Hardware-Specific Adaptations 205

A.1.4 Embedding Feasibility Results 207

A.2 vQPU Integration within HPC Schedulers 209

A.2.1 SLURM Generic Resource and Licenses Mechanism 209

A.2.2 Quantum-Partition Scheduling with HyperQueue 210

A.2.3 StreamFlow Quantum Connector 210

A.2.4 Circuit Cutting for Qubit-Scale Bridging 211

Appendix B Mathematical Formulations and Optimization Models **213**

B.1 Spectral Lower Bounds on the Chromatic Number 213

B.1.1 Hoffman Ratio Bound 213

B.1.2 Elphick–Wocjan σ-Bound 214

B.1.3 Edwards–Elphick Bound 214

B.1.4 Lovász Theta Function 215

B.1.5 Summary and Practical Notes 216

B.2 BBQ-mIS Branching Formalism 216

B.2.1 Maximal Independent Sets and the Coloring Theorem . . . 216

B.2.2 Branch-and-Bound Node State and Pruning 217

B.2.3 Exploration Priority Score 219

Appendix C Additional Experimental Results **220**

C.1 QRC Credit-Default Pipeline: Supplementary Details 220

C.1.1 Dataset Preprocessing Details 220
C.1.2 Aquila Register Embedding 222
C.1.3 Feature Correlation Analysis 222
C.1.4 DNN Architecture and Hyperparameters 223
C.2 QEK Graph-Classification Pipeline: Supplementary Details 225
C.2.1 PROTEINS Dataset Statistics 225
C.2.2 Waveform Comparison: Λ_a versus Λ_{BO} 227
C.2.3 Bayesian Optimization Configuration 228
C.3 BBQ-mIS: Supplementary Benchmark Results 230
C.3.1 Emulation Runtimes on the Small-Graph Benchmark 230
C.3.2 Large-Graph Benchmark with Classical MIS Surrogate . . . 231
C.4 Resource Management: Supplementary Experimental Results 232
C.4.1 Leonardo Production Results: Dynamic Allocation Strategies 232
C.4.2 Node Occupancy on Leonardo during the Experimental Campaign 234

List of Figures

3.1 Detailed hybrid pipeline for classification with QRC, including preprocessing, resampling, quantum encoding, time evolution, measurement, and classical classification. 46

3.2 Hybrid quantum-classical pipeline for graph classification via QEK. 56

3.3 Schematic comparison of execution patterns for QRC (a), quantum kernels (b), and variational QML (c). 60

4.1 Visualization of the hybrid workflow of the BBQ-mIS algorithm. Green boxes represent the classical parts of the workflow, while purple ovals the quantum ones. The graph coloring problem is decomposed into MIS subproblems solved via a quantum solver, while a classical Branch-and-Bound layer orchestrates the global search. 72

4.2 Representation of the Branch-and-Bound scheme underlying the BBQ-mIS algorithm. 75

4.3 Schematic comparison of execution patterns for hybrid optimization: (a) variational tuning loops (QAOA-like), (b) throughput-oriented annealing sampling and (c) quantum-assisted exact/decomposition methods with irregular, state-dependent quantum invocations. 79

5.1 Sequential execution of two hybrid jobs in a multi-tenant, single QPU scenario on a hybrid HPC–QC cluster. Job 2 cannot begin its classical phase until Job 1 releases the QPU, leaving both the QPU and classical nodes idle for extended intervals. 86

5.2 Time scales in hybrid QC–HPC execution (a) and related scheduling pathologies of naive co-allocation in hybrid jobs (b). 87

5.3 Reference architecture for exposing quantum resources to batch schedulers. 88

5.4 Time-based multiplexing with vQPUs: both jobs start concurrently; quantum phases are serialized by the device FIFO, inducing a queue overhead that is amortized when classical and quantum times are imbalanced. 93

5.5 Example of a (1,5)-clustered graph used as BBQ-mIS input (a) and the corresponding 9-qubit QAOA circuit with an identified circuit-cutting gate (b). The cut-move gate divides the circuit into two 5-qubit sub-circuits executed separately and recombined classically, without requiring ancilla qubits at execution time. 95

5.6 Quantum occupancy, quantum time, and mean queue time as functions of the number of concurrent replicas n_{copies}, for different quantum–classical workload ratios R. 98

5.7 Total completion time and per-job runtime distributions under vQPUs, compared to the baseline policy, for varying n_{copies} and R. . . . 99

5.8 A Quantum-HPC workload executed as a loosely-coupled, independently scheduled workflow. 102

5.9 Representation of a workload execution featuring malleability properties to handle the release and retrieval of resources when switching from classical to quantum-oriented workloads. 103

5.10 Hybrid clustering aggregation pipeline. The workflow alternates parallel classical clustering steps with a QUBO/MIS aggregation step suited for quantum or quantum-emulated resources, and a silhouette-based evaluation step driving the loop termination. 106

5.11 Timeline of cumulative classical node usage for two concurrent workloads under baseline, workflow, and malleability. The plot highlights how dynamic strategies reduce idle reservations and enable interleaving of classical phases. 110

5.12 Conceptual positioning of the main execution strategies for hybrid QC–HPC workloads. The figure compares vQPUs, workflow decomposition, and malleability along two dimensions: the time-scale granularity at which the quantum phase is managed and the degree to which the programming model is transparent to the user or explicitly exposed at application level. 112

6.1 PCA explained variance per component, computed on the `CARDS_30000` dataset. Components are ordered by descending explained variance; 11 components are retained, together explaining approximately 99% of the total variance. 125

6.2 Correlation circle for the 24 pre-processed features of `CARDS_30000`, projected onto the first two principal components. Features sharing a colour belong to the same agglomerative cluster; the 12 resulting clusters are subsequently averaged to form the 12-dimensional encoded input to the reservoir layer. 125

6.3 Example pulse sequence for the (a) position encoding and (b) detuning encoding. In the latter case, we considered a short 5 qubit array for the sake of readability. The relative local detuning waveforms are plotted using different colors. 128

6.4 Layered QEK time evolution: alternating application of the graph-dependent Hamiltonian $\hat{H}_G$ and the mixing Hamiltonian $\hat{H}_\theta$, parameterised by the pulse set Λ. 131

6.5 Example of original graph (left) and corresponding feasible UD embedding under hardware constraints (right). 133

6.6 Example QEK energy distributions for two graphs from the `PROTEINS12` dataset. Sub-figures (a) and (b) show the interaction-energy histogram alongside the corresponding graph structure; sub-figure (c) overlays the distributions of two graphs from different classes, illustrating the discriminative structure exploited by the Jensen–Shannon kernel. 135

6.7 QEK kernel matrix for PROTEINS12 (emulation, Λ_{BO}, $\mu = 1$). Each cell (i, j) encodes the Jensen–Shannon-based similarity between graphs G_i and G_j; block structure indicates class-related groupings. 136

6.8 Graph coloring results (number of colors used) for Greedy-it-MIS and BBQ-mIS on the 120-instance UD graph benchmark ($n \in \{10, \ldots, 15\}$ vertices, emulation via Pulser), compared with the Gurobi exact solver. BBQ-mIS always matches the Gurobi optimum; Greedy-it-MIS produces suboptimal colorings in 38 of 120 cases. 140

6.9 Absolute number of mISs sampled by the MIS solver and the relative percentage of the total, as a function of the total number of mISs and graph size, for shot counts $n \in \{2^6, \ldots, 2^{13}\}$ (emulation only). Results are averaged over ten independent runs per graph instance. . 143

6.10 Classification benchmark on CARDS_30000 (noiseless emulation): emulated QRC with detuning encoding, Classical Reservoir Computing (CRC), and Fully Classical (FC) pipelines compared against the DNN. The shaded band shows the DNN standard deviation; the red line marks the maximum F1-score across all compared methods. 144

6.11 QRC credit-default F1-score as a function of the number of Hamiltonian simulations (shot count) on CARDS_30000 (detuning encoding). Dotted lines show the corresponding noiseless statevector emulation scores; solid lines show shot-based emulation. The score saturates after a few hundred shots, indicating limited sensitivity to the sampling budget in this range. 145

6.12 Classification benchmark on CARDS_1000: real Aquila hardware and emulated QRC (both detuning and position encoding), CRC, and FC compared against the DNN. The shaded band shows the DNN standard deviation; the red line marks the maximum F1-score across all methods. Hardware results show notable degradation relative to noiseless emulation, with detuning encoding more robust than position encoding. 146

6.13 Classification benchmark on CARDS_2500: real Aquila hardware and emulated QRC (detuning encoding), CRC, and FC compared against the DNN. The shaded band shows the DNN standard deviation; the red line marks the maximum F1-score. The hardware-to-emulation gap is smaller than on CARDS_1000, indicating that the downstream classifier partially absorbs noise with more training data. 147

6.14 Comparison between emulated and Aquila hardware energy distributions for a randomly selected graph from PROTEINS12. Hardware execution broadens the distribution relative to noiseless emulation, consistent with decoherence and readout noise. 151

6.15 Hamiltonian definition for Aquila's tasks. For each graph, the Hamiltonian is defined by the lattice, the Rabi frequency and the detuning. The phase is always kept constant at zero. 156

6.16 Comparison of solution histogram between emulation on classical device and Aquila. In Figure 6.16a, 6.16b and 6.16c emulated solutions with most shots are prioritized, while in Figure 6.16d, 6.16e and 6.16f Aquila solutions with most shots are prioritized. For *graph_1* we consider a total of 892 shots, for *graph_2* 932 shots, and for *graph_3* 971 shots. 157

6.17 Comparison of solution distributions (ranked probability across all sampled bitstrings) between Bloqade emulation and Aquila hardware for the three BBQ-mIS subgraphs. Top row: emulation; bottom row: Aquila. Aquila results show broader distributions consistent with decoherence, imperfect state preparation, and readout noise. 158

6.18 Comparison of absolute number of mISs (a) and relative percentage of the total (b) recovered from Bloqade emulation and Aquila hardware for the three BBQ-mIS subgraphs (graph_1, graph_2, graph_3), using 128-shot batches. 159

A.1 DEN optimisation pipeline. The preprocessing phase initialises vertex coordinates using either the Scaling or Fruchterman-Reingold method. The learning phase trains the DEN to find a feasible UDG embedding while maximising the adjacency gap. The best feasible embedding found across all training epochs is returned. 203

A.2 Modified DEN architecture for Aquila. The *DistL* output layer is extended to $2\binom{n}{2}$ elements: the first $\binom{n}{2}$ are full squared Euclidean distances (fed to $\mathrm{ELF}_{\mathrm{min}}$ and $\mathrm{ELF}_{\mathrm{max}}$); the remaining $\binom{n}{2}$ are squared y-axis differences (fed to $\mathrm{ELF}_{\mathrm{row}}$). Example shown for a 5-vertex graph. 206

A.3 Percentage of PROTEINS graph samples with less than 16 nodes for which the DEN found an Aquila-feasible embedding, grouped by vertex count n. Numbers above each bar indicate the total samples in that bin. 208

C.1 Aquila register embedding strategies for a 1D atom array. (a) Single-column layout; (b) serpentine layout for longer chains; (c) replicated column with six copies; (d) replicated serpentine with four copies. Cross-replica separation of $15\,\mu$m suppresses inter-replica Rydberg interactions. 223

C.2 Statistical correlation between Aquila QRC features (detuning encoding) and shot-based emulation, as a function of the number of shots and timestep index, for ten randomly selected CARDS_2500 samples. Correlation degrades at later timesteps due to decoherence accumulation and saturates beyond ≈ 200 shots per timestep. 224

C.3 Node and edge distributions across the full PROTEINS dataset. Most graphs contain fewer than 100 nodes. The vertical dashed line at 12 nodes marks the boundary of PROTEINS12, used for classical emulation and pulse optimization; the boundary of PROTEINS256 at 256 nodes encompasses nearly the full dataset by count. 226

C.4 Pulse waveforms for the two QEK parameter sets. Both consist of three Rabi-drive segments (mixing Hamiltonian $\hat{H}_\theta$) separated by free-evolution intervals (Ising Hamiltonian $\hat{H}_G$). The optimized schedule Λ_{BO} is characterized by a dominant central pulse ($\approx 80^\circ$ rotation) flanked by two shorter pulses ($\approx 45^\circ$ and $\approx 18^\circ$), compared to the more uniform pulse durations of Λ_a. 227

C.5 Comparison of energy distributions produced by Λ_a (light red histogram labeled as "Pasqal") and Λ_{BO} (blue histogram labeled as "Optimized") on the same graph from PROTEINS12. The optimized schedule yields a distribution with greater separation between energy levels, producing a more discriminative kernel. 228

C.6 Graph coloring results on the large-graph dataset ($n \in \{20, \ldots, 100\}$) using a classical MIS surrogate in place of the quantum solver. BBQ-mIS solutions are at least as good as the benchmark across all tested instances. Greedy-it-MIS consistently returns colorings with more colors than necessary. 232

C.7 Node occupancy on the Leonardo DCGP partition over the experimental period (13–17 November 2024, 2-hour resolution). The cluster operates at near-full capacity for most of the period, with short troughs during off-peak hours. This persistent high load is the primary driver of the scheduling delays and elevated variance observed in the Leonardo experiments (Tables C.7 and C.8). 234

C.8 Node occupancy on the Leonardo Booster partition over the same period. The Booster partition (GPU-accelerated) shows a similar high-utilization pattern, confirming that Leonardo operated under substantial multi-tenant load throughout the experimental campaign. 235

List of Tables

2.1 Comparison of workload characteristics for QML and combinatorial optimization on NISQ devices. 25

3.1 Qualitative comparison of NISQ-relevant QML paradigms from an algorithmic and infrastructure perspective. 63

3.2 Infrastructure stressors and practical optimization knobs for NISQ QML paradigms. 64

4.1 Qualitative comparison of hybrid paradigms for combinatorial optimization in the NISQ era. 83

4.2 Infrastructure stressors and practical optimization knobs for hybrid combinatorial optimization workloads. 84

5.1 Comparison between Baseline, Workflow and Malleability execution types for hybrid workloads with 2 minutes quantum jobs. The table shows results for a single workload (Single Mode), and two concurrent copies (Double Mode). 109

5.2 Comparison between Baseline, Workflow and Malleability execution types for hybrid workloads with short ($<$ 1 second) quantum jobs. The table shows results for a single workload (Single Mode), and two concurrent copies (Double Mode). 111

5.3 Comparative summary of the three hybrid QC–HPC execution strategies. Rows capture the dimensions most relevant to deployment decisions; no single strategy dominates across all dimensions. . . . 115

6.1 Hyperparameter search ranges for the six lightweight classifiers evaluated in the QRC credit-card default study. CW: class weights enabled during training. 129

6.2 Composition of the two filtered subsets of the PROTEINS benchmark used for graph classification via QEK. PROTEINS12 (graphs with $\leq$12 nodes) is used for emulation and pulse optimisation; PROTEINS256 (graphs with $\leq$256 nodes, Aquila-embeddable) is used for hardware evaluation. 132

6.3 Training times (seconds) for the classifiers evaluated in the QRC credit-card default study on CARDS_30000, across resampling strategies. The DNN is trained on the original 12 preprocessed features; all other classifiers use the 390-dimensional QRC output features (detuning encoding, 5 timesteps). Columns: no resampling (None), SMOTE, K-Means SMOTE (kSMOTE), Cluster Centroids undersampling (CC). Values reported as mean $\pm$ standard deviation over five runs. 148

6.4 QRC experimental configurations and key results for credit-card default prediction. Execution modes: *SV* = noiseless statevector emulation; *Shot* = shot-based emulation; *HW* = real Aquila hardware. Qubits: 12 (detuning encoding) or 13 (position encoding); 5 timesteps; 2.5 μs total analog evolution. F1-scores reported as best mean value across classifiers and resampling strategies. 149

6.5 Classification performance on PROTEINS12 for QEK with parameter sets Λ_a and Λ_{BO} under emulation and real Aquila hardware execution, compared with the Shortest Path Kernel (SPK) baseline. . 149

6.6 QEK (Aquila hardware, Λ_{BO}) and SPK classification performance on PROTEINS256 (graph classification benchmark, 276 protein graphs). 151

6.7 QEK classification performance on PROTEINS256 (Aquila hardware) as a function of the number of shots used per graph for energy-distribution estimation. 152

6.8 BBQ-mIS hardware evaluation on Aquila: shot configuration for the three representative MIS subproblems extracted from a 20-node graph-coloring instance. Nominal budget is 1,000 shots per task; valid shots are those retained after discarding failed initializations. The 128-shot batch size is used for both the emulation (ten independent runs) and the hardware (pooled and partitioned) mIS recovery comparison. 159

6.9 Cross-case-study summary: workload patterns, quantum component roles, dominant bottlenecks, and key empirical findings across the three hybrid quantum–classical use-cases evaluated in this Chapter. The bottleneck and finding columns reflect conditions at the time of the experimental campaign (remote cloud-like QPU access, finite shot budgets, asynchronous submission). 168

A.1 DEN architecture for a graph with n vertices embedded in $N = 2$ dimensions. All fully connected layers include bias nodes and dropout (probability p_{drop}), except the fixed-weight layers. 204

B.1 Spectral lower bounds on the chromatic number $\chi(G)$ used in BBQ-mIS. All are computable in polynomial time. The tightest bound is used at each Branch-and-Bound node. 216

C.1 Feature clusters produced by agglomerative clustering in PCA score space on `CARDS_30000`. Each cluster becomes one input feature for the QRC layer after averaging. 221

C.2 Training-set class distributions after each resampling strategy, for the three dataset scales. Validation and test sets are held fixed (class-1 counts in parentheses). CC = Cluster Centroids; K-SMOTE = K-Means SMOTE. 221

C.3 DNN architecture used as the classical nonlinear baseline. Input dimensionality $f = 12$ (preprocessed features). All hidden layers include dropout regularization. The output layer uses softmax activation for binary classification. 225

C.4 Overall statistics of the `PROTEINS` dataset before filtering. 226

C.5 Comparison of the two QEK pulse parameter sets. Rotation angles are computed as $\theta_i = \Omega_0 \tau_i$ with $\Omega_0 = 15.8\,\mathrm{rad}/\mu\mathrm{s}$. The Λ_{BO} schedule is 37% shorter in total duration, reducing decoherence exposure. 227

C.6 Wall-clock emulation time (hh:mm:ss) for Greedy-it-MIS and BBQ-mIS on the small-graph benchmark (Pulser-based quantum MIS solver). Times increase with n due to the exponential cost of state-vector simulation. BBQ-mIS is slower than Greedy-it-MIS because it invokes the MIS solver at multiple Branch-and-Bound nodes; the overhead diminishes in relative terms as the first (largest) MIS call dominates. 230

C.7 Leonardo: executions with two-minute-long quantum jobs (10 valid runs out of 32 scheduled). Wall time and node-second resource usage are averaged over valid runs; standard deviations reflect multi-tenant cluster conditions. 233

C.8 Leonardo: executions with short (< 1 second) quantum jobs (29 valid runs out of 32 scheduled). High standard deviations reflect multi-tenant scheduling interference during the experimental period. 234

List of Abbreviations

API Application Programming Interface

BBQ-mIS Branch-and-Bound Quantum maximal Independent Set solver

COBYLA Constrained Optimization by Linear Approximations

DAG Directed Acyclic Graph

DD Dynamical Decoupling

DEN Distance Encoder Networks

FIFO First-In First-Out

FPGA Field-Programmable Gate Array

GPU Graphics Processing Unit

HPC High-Performance Computing

MIS Maximum Independent Set

MPI Message Passing Interface

NISQ Noisy Intermediate-Scale Quantum

PEC Probabilistic Error Cancellation

PQC Parameterized Quantum Circuit

QAA Quantum Adiabatic Algorithm

QAOA Quantum Approximate Optimization Algorithm

QC Quantum Computing

QEK Quantum Evolution Kernel

QML Quantum Machine Learning

QNN Quantum Neural Network

QPU Quantum Processing Unit

QRC Quantum Reservoir Computing

QUBO Quadratic Unconstrained Binary Optimization

RC Reservoir Computing

SA Simulated Annealing

SLURM Simple Linux Utility for Resource Management

SPSA Simultaneous Perturbation Stochastic Approximation

SVM Support Vector Machine

UD Unit-Disk

VQA Variational Quantum Algorithm

VQC Variational Quantum Circuit

VQE Variational Quantum Eigensolver

ZNE Zero-Noise Extrapolation

mIS maximal Independent Set

vQPU Virtual Quantum Processing Unit

Chapter 1

Introduction

Quantum computing has emerged as one of the most disruptive technological paradigms of the last decades. By exploiting uniquely quantum mechanical phenomena such as superposition and entanglement, quantum processors promise computational advantages for classes of problems that are intractable for classical computers. Nevertheless, despite significant theoretical breakthroughs and rapid hardware development, current quantum devices remain limited in scale, coherence time, and error rates. These constraints define what is commonly referred to as the Noisy Intermediate-Scale Quantum (NISQ) era.

In this regime, fully fault-tolerant quantum computation is not yet available. Instead, practical quantum applications must operate under severe hardware constraints: limited qubit counts, restricted connectivity, short coherence times, and non-negligible gate and readout errors. As a consequence, purely quantum solutions are rarely feasible. Instead, most practical approaches rely on hybrid quantum-classical algorithms, where quantum processors are embedded within classical optimization or control loops.

This hybrid paradigm reshapes the role of quantum computing. Rather than replacing classical high-performance computing (HPC) systems, quantum devices act as accelerators for specific computational kernels, such as state preparation, sampling, or expectation value estimation. The bulk of the workload, including data preprocessing, optimization, orchestration, and post-processing, remains classical. This division of responsibilities introduces new challenges that go far beyond algorithm design.

Indeed, hybrid execution models require tight interaction between heterogeneous computational layers: classical CPUs and GPUs, quantum processing units (QPUs), network interfaces, compilers, runtime systems, and resource managers. The performance of a hybrid algorithm does not depend solely on circuit depth or qubit count, but also on latency, queueing policies, data transfer overhead, scheduling strategies, and orchestration mechanisms. As quantum resources are scarce and often remotely accessed, efficient integration with HPC infrastructures becomes a critical requirement.

The problems that arise from this integration are concrete and recurrent, and they motivate the present work. A variational loop that spends milliseconds executing on a QPU may wait minutes in a remote provider queue, so that queueing delay, rather than computation, dominates end-to-end runtime; this effect is referred to as latency amplification. Prevailing access models, based on cloud queues or exclusive batch reservations, force either the quantum or the classical side of an application to sit idle while the other works. Hardware topologies and native operations constrain which problem instances can be embedded on a device at all. Finally, a cluster may host thousands of classical cores alongside a single QPU, and naive co-allocation of both fragments the classical partition while leaving the scarcest resource underutilized. In current practice, these issues determine whether hybrid methods are usable at least as much as gate fidelity does.

This dissertation addresses these challenges from a full-stack perspective. Rather than focusing exclusively on algorithmic innovation or hardware characterization, this work investigates how hybrid quantum-classical methods can be implemented, optimized, and executed efficiently across all layers of the computational stack, from abstract models to physical deployment platforms.

The central thesis of this work is that, in the NISQ era, meaningful quantum advantage cannot be pursued in isolation at the algorithmic level. Instead, it must emerge from the co-design and co-optimization of algorithms, software abstractions, execution models, and platform-level resource management strategies. This holistic viewpoint is particularly relevant for application domains such as Quantum Machine Learning (QML) and combinatorial optimization, where hybrid iterative workflows dominate.

1.1 Hybrid Quantum–Classical Computing in the NISQ Era

The NISQ paradigm describes the current generation of quantum processors characterized by tens to a few hundreds of physical qubits, limited connectivity, and the absence of full quantum error correction. These devices enable experimental demonstrations of quantum algorithms but impose strict limitations on circuit depth and computational reliability.

Within this context, hybrid quantum-classical algorithms have emerged as the most promising strategy for near-term applications. Prominent examples include the Variational Quantum Eigensolver (VQE), the Quantum Approximate Optimization Algorithm (QAOA), quantum kernel methods, and various parametrized quantum circuits used in QML. These approaches share a common structure: a parameterized quantum circuit is executed on a QPU, measurement outcomes are collected, and a classical optimizer updates the parameters iteratively.

This iterative control loop fundamentally changes the computational model. A single logical algorithm may require hundreds or thousands of quantum circuit executions. Each execution involves classical-to-quantum communication, job submission to a QPU backend, queue waiting time, circuit transpilation, hardware execution, and measurement readout. Consequently, execution latency and resource contention become as relevant as quantum gate fidelity.

Furthermore, quantum resources are typically scarce. In most HPC–QC integration scenarios, a cluster may contain hundreds or thousands of classical cores but only one or a few QPUs. This asymmetry creates a strong imbalance in resource availability. If not properly managed, the QPU becomes a bottleneck, degrading overall system throughput and user experience.

In addition to hardware constraints, hybrid workloads introduce structural challenges of their own. Quantum kernels are invoked repeatedly inside classical optimization loops, and the two sides must synchronize at fine granularity, so classical execution may stall while waiting for quantum results. When QPUs are accessed remotely, queueing introduces delays that are difficult to predict and often dominate the iteration time. Moreover, classical and quantum tasks follow different resource allocation models, which complicates their joint scheduling. These characteristics make

hybrid applications fundamentally different from traditional HPC workloads. While classical HPC applications are often compute-intensive and throughput-oriented, hybrid QC–HPC applications are interaction-intensive and latency-sensitive.

Two major application domains exemplify these dynamics and motivate this dissertation. The first is Quantum Machine Learning, where parameterized quantum circuits are embedded within classical training loops. Training requires repeated evaluation of quantum circuits over multiple data samples, often with gradient estimation procedures. This leads to a large number of short quantum executions interleaved with classical optimization steps. The second is combinatorial optimization, where hybrid strategies use quantum subroutines, such as QAOA-based solvers or sampling procedures, within classical frameworks, including branch-and-bound or decomposition methods. In large-scale problems, quantum components may represent only a fraction of the total computation but critically influence convergence behavior.

In both domains, the efficiency of hybrid execution depends only in part on quantum algorithm design. It depends equally on how circuits are batched before submission, how classical-quantum communication overhead is contained, how scarce QPU time is scheduled among competing tasks, and how the resulting workloads are integrated with HPC batch schedulers and orchestrated at the workflow level.

Therefore, addressing hybrid computing in the NISQ era requires rethinking the entire computational stack. It is insufficient to optimize circuits in isolation; one must consider deployment environments, resource management policies, and execution models.

1.2 Full-Stack Perspective: From Algorithms to Platforms

The term *full-stack* in the context of this dissertation refers to the vertical integration of all layers involved in hybrid quantum-classical computation. At the top, the algorithmic layer concerns the design of hybrid quantum algorithms for QML and optimization. Below it, the software abstraction layer provides programming models, circuit representations, and classical-quantum interfaces. The runtime and execution layer governs iterative control loops, batching strategies, and invocation patterns. The

resource management layer implements scheduling mechanisms and virtualization strategies. Finally, the platform layer concerns the physical deployment of QPUs within HPC infrastructures.

Traditional research often focuses on one of these layers independently. However, hybrid NISQ workloads expose strong cross-layer dependencies. The depth and structure of a variational circuit influence execution time and queue occupancy. The batching strategy adopted at runtime affects QPU utilization and the extent to which latency can be amortized. The resource allocation policy determines whether classical processes remain idle while waiting for quantum results. The presence of virtualization mechanisms, in turn, impacts throughput and fairness among users. A cross-layer perspective enables coordinated optimization across these dimensions.

In this thesis, such an approach is applied in two complementary directions. The first direction proceeds from algorithms to infrastructure. Starting from the QML and combinatorial optimization use-cases, this work analyzes how their structural properties impact infrastructure-level behavior: iterative training loops generate bursty quantum workloads, decomposition-based optimization creates heterogeneous job sizes, and hybrid branch-and-bound introduces dynamic task generation. The second direction proceeds from infrastructure back to algorithms, since platform constraints inform algorithm design choices. Limited QPU availability motivates circuit batching, queueing latency encourages asynchronous invocation, and hardware topology influences problem encoding strategies.

This bidirectional interaction culminates in two key architectural contributions explored in later Chapters. The first is the Virtual Quantum Processing Unit (vQPU), a virtualization layer enabling time-based multiplexing of quantum resources within HPC environments. The second is a set of platform-aware resource management strategies, comprising workflow-aware scheduling and malleability concepts tailored to hybrid QC–HPC applications.

In addition to software and orchestration contributions, this dissertation also addresses practical deployment aspects, including the integration of an on-premises quantum system within an academic HPC environment. This real-world perspective ensures that the proposed methods are validated beyond simulation or purely theoretical settings.

Ultimately, the cross-layer approach adopted in this work aims to bridge the gap between algorithmic promise and practical execution. In the NISQ era, where hard-

ware limitations are unavoidable, extracting meaningful performance improvements requires careful coordination across abstraction layers.

The contributions of this dissertation demonstrate that hybrid quantum-classical computation must be treated not merely as an algorithmic problem, but as a systems-level challenge, where efficiency, scalability, and usability depend on the coherent integration of algorithms, software, and platforms.

1.3 Research Questions and Thesis Contributions

The overarching goal of this dissertation is to investigate how hybrid quantum-classical methods can be efficiently implemented and executed on NISQ devices when integrated within High-Performance Computing environments. This objective is pursued through a cross-layer methodology, spanning from algorithmic modeling to platform-level resource management.

Four research questions (RQs) structure this investigation. They are not independent of one another. Each arises from the problems left open by the previous one, tracing a path that ascends the computational stack from algorithm design to platform-level resource management. The same path organizes the chapters that follow, and Chapter 8 returns to each question with the evidence gathered along the way.

RQ1: How can hybrid quantum-classical algorithms for Quantum Machine Learning and combinatorial optimization be designed and implemented to operate effectively under NISQ constraints? The research path begins at the algorithmic layer, because NISQ limitations in qubit count, noise, and connectivity decide which hybrid methods are feasible before any infrastructure concern arises. This thesis analyzes different paradigms within QML (including variational models, reservoir computing, and quantum kernels) and hybrid optimization (including QUBO formulations and variational quantum optimization), evaluating their structural properties, scalability limits, and suitability for real hardware deployment. Particular attention is devoted to understanding how algorithmic design choices propagate downward to execution and infrastructure layers.

RQ2: What execution models best capture the behavior of hybrid quantum-classical workloads in HPC environments? The algorithmic analysis conducted for RQ1 shows that design choices manifest at run time as distinctive execution patterns, which existing HPC workload models do not capture. Hybrid algorithms are inherently iterative and interaction-intensive. Unlike traditional HPC applications, they require repeated quantum invocations within classical control loops, introducing latency, synchronization barriers, and queueing effects. This research question investigates how such workloads can be formally characterized in terms of execution patterns, control-loop structures, and resource interaction models. The analysis highlights the impact of latency, batching strategies, and classical-quantum communication overhead on overall performance, providing a conceptual framework for platform-aware optimization.

RQ3: How can scarce quantum resources be efficiently integrated and virtualized within HPC infrastructures? Characterizing hybrid execution exposes a structural bottleneck: the asymmetry between abundant classical resources and scarce QPUs. Efficient integration of the two therefore becomes a central challenge. This question addresses the design of a virtualization layer capable of exposing quantum resources to HPC schedulers in a controlled and scalable manner. The vQPU abstraction introduced in this thesis enables time-based multiplexing of quantum hardware, improving overall utilization while maintaining transparency for user applications. The architectural design, scheduling model, and performance characterization of this virtualization layer are investigated both analytically and experimentally.

RQ4: Which platform-aware resource management strategies improve the performance and scalability of hybrid QC–HPC workloads? Beyond virtualization, hybrid workloads require scheduling policies that account for their iterative and heterogeneous nature; otherwise, classical nodes sit idle during quantum phases and cluster utilization degrades. This question explores static versus dynamic allocation strategies, workflow-aware planning, and malleability concepts for hybrid applications. The goal is to determine how resource management policies influence makespan, throughput, fairness, and utilization in production-like environments.

In answering these research questions, this dissertation provides five main contributions. The first is a systematic analysis of hybrid QML and combinatorial optimization paradigms under realistic NISQ constraints, supported by multiple case studies. The second is a formal characterization of hybrid execution patterns, highlighting latency and synchronization effects in HPC-integrated scenarios. The third is the design and implementation of the vQPU abstraction layer, enabling quantum resource virtualization and integration with HPC schedulers. The fourth is the development and evaluation of platform-aware resource management strategies, including workflow-level optimization. The fifth is an experimental validation conducted on both simulated and real infrastructures, including the deployment of an on-premises quantum system within an academic HPC environment.

Collectively, these contributions demonstrate that performance improvements in the NISQ era emerge not from isolated algorithmic advances, but from coordinated cross-layer optimization across the entire computational stack.

1.4 Structure of the Thesis

The remainder of this dissertation follows the research path traced by the four questions above, moving from foundations, through algorithms and execution models, to experimental validation and conclusions.

Chapter 2 provides the necessary background and reviews the state of the art. It introduces NISQ hardware constraints, variational and hybrid quantum algorithms, Quantum Machine Learning techniques, and quantum approaches to combinatorial optimization. It also discusses classical HPC systems, batch scheduling mechanisms, and the challenges inherent to QC–HPC integration, from which the research gaps addressed by this thesis are derived.

Chapter 3 takes up the algorithmic side of RQ1, focusing on Quantum Machine Learning on NISQ devices. It analyzes different QML paradigms, including variational quantum models, Quantum Reservoir Computing, and quantum kernel methods. Two case studies are presented, followed by a comparative discussion and an examination of the infrastructure-level implications of QML workloads.

Chapter 4 completes the treatment of RQ1 by addressing hybrid quantum approaches to combinatorial optimization. After introducing QUBO and Ising formula-

tions, the Chapter examines variational quantum optimization and hybrid classical-quantum cooperation strategies. One case study is presented, culminating in a comparative and infrastructure-aware analysis of the considered approaches.

Based on the execution models characterized in response to RQ2, Chapter 5 investigates platform-aware execution and resource management strategies for hybrid QC–HPC systems, addressing RQ3 and RQ4. It unifies virtualization-based and scheduling-based approaches within a coherent framework aimed at improving performance, scalability, and resource utilization in the NISQ era. The Chapter first introduces the vQPU architecture for quantum resource virtualization and its integration with HPC schedulers such as SLURM. Complementary strategies are analyzed, including workflow-aware planning, static versus dynamic allocation policies, and malleability for hybrid workloads. A comparative analysis highlights the trade-offs among these strategies across different hardware targets and execution scenarios.

Chapter 6 presents the experimental evaluation of the case studies outlined in Chapters 3 and 4. It describes the platforms and testbeds used, defines methodologies, and reports results for both QML and optimization use-cases, including scalability and utilization analyses.

Chapter 7 provides a comprehensive discussion of lessons learned from both the algorithmic and infrastructural perspectives. It reflects on the implications of cross-layer co-design in the NISQ era, outlines guidelines for production-grade hybrid QC–HPC systems, and discusses future directions beyond current hardware limitations.

Chapter 8 concludes the dissertation and closes the research path opened here. It revisits each research question with the evidence gathered, summarizes the main contributions alongside their empirical outcomes, identifies the principal limitations of the work, and outlines concrete directions for future research.

Finally, the appendices provide implementation details, mathematical formulations, and additional experimental results supporting the main text.

Chapter 2

Background and State of the Art

This Chapter provides the technical and conceptual background needed for the rest of the dissertation. The discussion begins with the main characteristics and limitations of current NISQ devices, with particular attention to the constraints that most directly affect the design and execution of hybrid quantum–classical methods. It then reviews the main algorithmic paradigms that have emerged in this regime, spanning variational methods, Quantum Machine Learning, and quantum approaches to combinatorial optimization. Since the focus of this thesis is not only algorithmic but also infrastructural, the Chapter further introduces the main concepts of High-Performance Computing systems, batch scheduling, and the challenges associated with integrating scarce quantum resources into classical computing environments. The goal is to position the thesis with respect to the current state of the art and to identify the research gaps addressed in subsequent Chapters.

2.1 NISQ Devices and Hardware Constraints

The current NISQ era, a term introduced by Preskill [17], is characterised by quantum devices comprising tens to a few hundred qubits, limited coherence times, imperfect gate fidelities, and the absence of full quantum error correction. Despite rapid hardware improvements, including increased qubit counts and improved gate fidelities, current systems remain fundamentally noise-limited [18, 19]. A systematic understanding of the physical origins of these limitations, the noise channels through which they manifest, the mitigation strategies available to practitioners, and the

compilation overheads they introduce is essential for designing algorithms that can be executed effectively on present devices. This section reviews each of these aspects in turn and positions them with respect to the experimental platforms employed in this thesis.

2.1.1 Hardware Modalities and Architectural Trade-Offs

Leading quantum hardware platforms include superconducting qubits [18], trapped ions [20], neutral atoms [21–23], and photonic architectures [24]. Each platform exhibits a distinct balance of gate speed, coherence time, connectivity, and control complexity, which determines the class of circuits it can support at acceptable fidelity.

Superconducting qubits. Superconducting transmon qubits are fabricated using Josephson junction circuits and operated at millikelvin temperatures. Single-qubit gate times are of order 10–50 ns, while two-qubit gate times typically fall between 100 and 500 ns. Coherence times T_1 and T_2 reach up to several hundred microseconds on state-of-the-art devices, though they are susceptible to charge noise, flux noise, and two-level-system defects in substrate materials. Connectivity is fixed by on-chip wiring; IBM's heavy-hex lattice, for instance, limits each qubit to at most three neighbours. This sparse topology reduces unwanted crosstalk at the cost of increased routing overhead during circuit compilation.

Trapped ions. Trapped-ion systems confine ions in electromagnetic traps and encode qubit information in hyperfine or optical transitions. Coherence times can reach seconds to minutes, and two-qubit gate fidelities routinely exceed 99.5%. Entangling operations via the Mølmer–Sørensen interaction are significantly slower than superconducting gates, typically requiring 1–10 ms, but all-to-all connectivity eliminates SWAP-routing overhead. Shuttle-based multi-zone architectures are currently under development for scaling to larger qubit registers [20].

Neutral atoms. Neutral-atom arrays use optical tweezers to arrange atoms in programmable two-dimensional configurations. Entangling operations are mediated by Rydberg blockade interactions, which are active only when two atoms are separated by less than the Rydberg radius r_b. This geometric constraint defines a

Unit-Disk (UD) graph structure: an edge is present between atoms i and j if and only if $|x_i - x_j| < r_b$. This property is central to the QEK and BBQ-mIS methods developed in this thesis, as it enables direct embedding of graph problems onto the hardware connectivity. Neutral-atom platforms support both analog (pulse-shaped Hamiltonian evolution) and digital (gate-based) operating modes; the Aquila device employed in this work operates in analog mode [21, 23].

Photonic architectures. Photonic processors encode quantum information in optical modes and operate at room temperature [24]. They are inherently robust to certain noise types but suffer from photon loss, probabilistic two-qubit gates under linear-optical schemes, and scalability constraints on deterministic photon sources. These limitations currently confine photonic processors to more specialised and speculative roles in near-term hybrid computing.

Hardware benchmarking methodologies have evolved substantially in recent years. Randomised benchmarking, cross-entropy benchmarking, and cycle benchmarking are now complemented by application-level benchmarks [25]. These metrics attempt to quantify effective algorithmic depth under realistic noise conditions, providing more informative cross-platform comparisons than gate-level fidelity figures alone.

2.1.2 Noise Channels and Decoherence Mechanisms

Noise in quantum circuits arises from a variety of physical processes. A systematic characterisation of the relevant channels is important both for designing noise-aware algorithms and for selecting appropriate mitigation strategies. The principal noise sources in NISQ devices are reviewed below.

Depolarizing noise. The depolarizing channel constitutes the standard first-order model for gate imperfections. It applies each of the three Pauli operators X, Y, Z with equal probability $p/3$ and leaves the state unchanged with probability $1-p$:

$$\mathscr{E}(\rho) = (1-p)\,\rho + \frac{p}{3}\,(X\rho X + Y\rho Y + Z\rho Z)\,.$$

Two-qubit depolarizing error rates on current superconducting hardware are typically an order of magnitude larger than single-qubit rates, as entangling gate implementations are more sensitive to control imperfections.

Amplitude damping. Amplitude damping models energy relaxation (T_1 decay), corresponding to spontaneous emission from the excited state $|1\rangle$ to the ground state $|0\rangle$. Its Kraus operators are:

$$K_0 = \begin{pmatrix} 1 & 0 \\ 0 & \sqrt{1-\gamma} \end{pmatrix}, \quad K_1 = \begin{pmatrix} 0 & \sqrt{\gamma} \\ 0 & 0 \end{pmatrix},$$

where $\gamma = 1 - e^{-t/T_1}$ is the decay probability over circuit execution time t. Deep circuits are particularly susceptible when their total duration approaches T_1.

Dephasing. Phase damping, or dephasing, models the loss of phase coherence without energy exchange. It is characterised by the coherence time T_2 (or the effective time T_2^* under inhomogeneous broadening) and destroys off-diagonal elements of the density matrix while leaving populations unchanged:

$$\mathscr{E}(\rho) = \left(1 - \frac{\lambda}{2}\right)\rho + \frac{\lambda}{2} Z\rho Z,$$

where $\lambda = 1 - e^{-t/T_2}$. Calibration drift in T_2 over extended variational runs can introduce systematic bias in expectation value estimates [26].

Crosstalk. Crosstalk arises from unwanted interactions between simultaneously driven qubits. In superconducting architectures, it manifests as frequency collisions between neighbouring qubits during parallel gate execution. Crosstalk is not captured by independent single-qubit noise models and requires multi-qubit characterisation protocols such as simultaneous randomised benchmarking [27]. Its magnitude depends on physical layout, qubit frequency assignment, and gate scheduling.

Readout errors. Measurement-induced errors result in qubit state misidentification. They are characterised by a confusion matrix $\mathscr{A}$ with elements $\mathscr{A}_{ij} = \Pr(\text{read } i \mid \text{prepared } j)$. Readout error rates of 1–5% per qubit are common on near-term

devices [28]. Correlated readout errors across multiple qubits further complicate characterisation and can bias multi-qubit expectation value estimates.

While standard noise models assume Markovian dynamics, recent studies highlight non-Markovian effects and temporal drift in real hardware [26, 28]. Temporal drift, slow variation of device parameters over hours, is particularly problematic for variational workloads that require consistent circuit performance across many evaluation episodes.

2.1.3 Error Mitigation Techniques

In the absence of full quantum error correction, software-level error mitigation techniques reduce the bias introduced by noise on expectation value estimates without requiring additional physical qubits. Unlike error correction, these techniques do not suppress errors at the physical level; instead, they restructure measurements or post-process results to produce less biased estimates, at the cost of increased sampling overhead [29].

Zero-noise extrapolation. Zero-noise extrapolation (ZNE) estimates the noiseless expectation value by intentionally amplifying the noise level and fitting an extrapolation model to the zero-noise limit [30]. Noise amplification is achieved either by gate folding, replacing each gate G with $GG^{\dagger}G$, or by pulse-level stretching. Richardson extrapolation and polynomial fitting serve as common extrapolation ansätze. ZNE requires no additional qubits and can be applied to existing circuits without structural modification, but its accuracy degrades for deep circuits where amplified noise pushes results outside the reliable extrapolation range.

Probabilistic error cancellation. Probabilistic error cancellation (PEC) decomposes the ideal quantum channel as a quasi-probability distribution over noisy implementable operations [31]. Sampling from this distribution with appropriate signs yields an unbiased estimator of the ideal expectation value. The sampling overhead grows exponentially in the number of noisy gates, making PEC expensive for deep circuits. Recent approaches reduce this overhead through error learning and sparse quasi-probability representations [29].

Measurement error mitigation. Readout errors can be partially corrected by inverting the calibration matrix $\mathscr{A}$, obtained by preparing each computational basis state and recording the observed outcome distribution. For n qubits, full calibration requires 2^n preparation circuits; scalable approximations assuming local or tensor-product readout noise are therefore employed in practice [28]. These corrections substantially reduce systematic bias in expectation values, particularly when per-qubit readout error rates are large.

Dynamical decoupling. Dynamical decoupling (DD) applies sequences of control pulses during qubit idle periods to average out low-frequency noise and extend effective coherence [32]. Standard sequences include the Hahn echo, CPMG, and XY-4. Unlike other mitigation methods, DD operates at the pulse level and does not require additional circuit shots; its effectiveness depends on the spectral density of the noise and the pulse timing resolution available on the backend.

Error mitigation methods are not mutually exclusive and are often combined in practice. However, their collective overhead can increase total circuit execution counts by factors of 10^2–10^3 [29]. This amplifies classical–quantum communication overhead and introduces additional latency in hybrid iterative loops, a consideration that directly motivates the infrastructure and scheduling strategies developed in Chapter 5.

2.1.4 Connectivity and Compilation Overheads

Circuit compilation translates a logical quantum circuit into a sequence of native hardware-executable operations, respecting the physical qubit connectivity of the target device. This process introduces two principal sources of overhead: gate decomposition and qubit routing.

Native gate sets. Each hardware platform exposes a physically realisable gate set. IBM superconducting devices use the set $\{R_Z, \sqrt{X}, X, \text{CNOT}\}$ (or the ECR gate on newer architectures), while trapped-ion processors support native Mølmer–Sørensen interactions. Any logical gate must first be decomposed into the native set before routing is applied, adding circuit depth even before connectivity constraints are enforced.

SWAP insertion. Modern superconducting architectures impose sparse nearest-neighbour connectivity. When a circuit requires a two-qubit gate between non-adjacent qubits, SWAP operations must be inserted to route qubit states to adjacent positions. Each SWAP gate decomposes into three CNOT gates, introducing three additional two-qubit error events per routing step. For circuits with high entanglement or large connectivity mismatch, SWAP insertion can double or triple the effective two-qubit gate count, with proportional increases in accumulated error.

Routing algorithms. Qubit routing is NP-hard in general; practical compilers rely on heuristic algorithms. The SABRE algorithm [33] uses a lookahead cost function to minimise SWAP insertion depth. Noise-adaptive transpilation [34] augments routing with real-time calibration data, preferentially routing critical gates through lower-error qubit pairs. Layout-aware scheduling [35] further reduces crosstalk by separating simultaneously executed two-qubit gates onto non-interacting edges.

Transpiler optimisation levels. Tools such as Qiskit expose multiple optimisation levels (0–3), applying progressively more aggressive gate cancellation, commutation analysis, and peephole optimisation passes. Higher levels reduce circuit depth at the cost of increased compilation time. For iterative variational workloads, the compilation strategy must balance depth reduction against compilation latency, particularly when device calibration data changes between experimental sessions.

For variational algorithms evaluated across thousands of parameter configurations, a common strategy is to compile the circuit once against a fixed template and bind parameters at runtime, avoiding per-iteration recompilation. However, this approach fixes the circuit structure and may not track changes in device calibration over time. Calibration drift, gradual degradation in circuit fidelity as device parameters shift, is a practical concern in long-running experiments and is further discussed in the context of the experimental campaigns reported in Chapter 6.

2.1.5 Toward Early Fault-Tolerant Systems

While this thesis focuses on NISQ systems, recent hardware milestones indicate the beginning of a transition toward early fault-tolerant quantum computing. Under-

standing these developments contextualises the current limitations of NISQ methods and clarifies the horizon beyond which they will be superseded.

Quantum error correction requires that physical error rates fall below a code-specific threshold [36]. Below this threshold, logical error rates can be suppressed exponentially by increasing the code distance. In 2023, Google reported the first experimental demonstration that increasing the distance of a surface code from $d = 3$ to $d = 5$ reduces the logical error rate, thereby crossing the error correction threshold [37]. This result, obtained on the Sycamore processor, constitutes a qualitative milestone: adding more physical qubits produced a better logical outcome for the first time.

IBM has demonstrated repeated syndrome extraction using flag-qubit protocols on heavy-hex devices [38], with a hardware roadmap targeting fault-tolerant logical qubits on processors exceeding 10^3 physical qubits within the current decade. Quantinuum's trapped-ion platform has achieved high-fidelity logical operations on small-distance codes, leveraging two-qubit gate fidelities exceeding 99.8% to approach fault-tolerant operation [39].

Despite these advances, resource estimates for practically relevant fault-tolerant computation remain several orders of magnitude beyond current hardware capabilities [40]. A single logical qubit suitable for a cryptographically relevant algorithm requires hundreds to thousands of physical qubits after accounting for syndrome measurement overhead and ancilla requirements. Resource estimates for 2048-bit RSA factorisation require on the order of 10^8 Clifford and T-gate operations [40], far exceeding the coherent depth achievable on any current device.

Hybrid and NISQ-compatible algorithms therefore remain the primary viable computational paradigm for the time frame of this dissertation. The practical utility of these methods should be assessed relative to computational tasks tractable within present hardware constraints, rather than against the theoretical promise of fault-tolerant quantum computation.

2.2 Variational and Hybrid Quantum Algorithms

Variational Quantum Algorithms (VQAs) represent the dominant algorithmic paradigm for NISQ devices [41]. They combine parameterized quantum circuits with classical optimization routines in a hybrid feedback loop.

2.2.1 General Structure

Given parameters θ, a circuit prepares:

$$|\psi(\theta)\rangle = U(\theta)|0\rangle,$$

and minimizes a cost function:

$$C(\theta) = \langle\psi(\theta)|H|\psi(\theta)\rangle.$$

The hybrid loop continues until convergence.

Some reviews emphasize that the classical optimizer often dominates total runtime due to repeated circuit evaluations [19, 41]. Moreover, optimizer choice (gradient-based vs gradient-free) strongly impacts shot efficiency and convergence stability.

More recent works have investigated several directions for improving this baseline scheme: the trade-off between expressibility and trainability [42], layerwise training strategies [43], adaptive ansätze construction as in ADAPT-VQE [44], and noise-resilient circuit architectures [45].

The barren plateau phenomenon [46] remains a central theoretical limitation. Recent analyses suggest that local cost functions and shallow problem-inspired circuits partially mitigate gradient concentration [47].

2.2.2 Shot Complexity and Resource Scaling

Shot noise significantly impacts gradient estimation. Recent theoretical bounds show that gradient variance scales unfavorably with system size [48]. As a consequence, practical VQAs may require thousands to millions of circuit evaluations.

This repeated invocation pattern creates execution bottlenecks when quantum hardware is accessed remotely or through batch systems.

2.2.3 Distributed and Parallel VQA Strategies

Emerging research explores distributed parameter evaluation and parallel shot allocation [49]. Such strategies are directly relevant for HPC-integrated quantum workflows, where classical nodes may evaluate different parameter batches concurrently.

These developments highlight that algorithmic design cannot be decoupled from execution infrastructure, a key thesis motivation.

2.3 Quantum Machine Learning in the NISQ Era

Quantum Machine Learning aims to combine quantum computation with data-driven modeling [50]. In the NISQ era, most practical QML models rely on variational circuits or quantum kernels. Recently, the QML field has matured from proof-of-concept demonstrations toward more critical evaluation of quantum advantage claims [51, 52]. Several studies question whether near-term QML models provide practical advantages over classical baselines.

Nonetheless, QML remains a valuable framework for studying hybrid execution models due to its iterative and data-intensive nature.

2.3.1 Data Encoding and Feature Maps

Encoding classical data into quantum states remains a central challenge. In [53] the authors analyze expressivity of quantum feature maps and their relation to kernel methods.

Quantum feature maps define kernels:

$$K(x,x') = |\langle \Phi(x) | \Phi(x') \rangle|^2,$$

which can be evaluated on quantum hardware [54]. However, dataset scaling requires $O(N^2)$ circuit executions, introducing substantial infrastructure overhead.

2.3.2 Trainability and Generalization

Trainability is a fundamental theoretical concern in variational QML, distinct from but related to expressibility and generalisation properties.

Barren plateaus. The barren plateau phenomenon, identified by McClean et al. [46], describes the exponential vanishing of cost function gradients as system size increases for generic parameterised circuits. For a global cost function on n qubits and a hardware-efficient ansatz of sufficient expressibility, the gradient variance satisfies:

$$\mathrm{Var}\left[\partial_{\theta_k} C(\theta)\right] \in \mathscr{O}\left(2^{-n}\right),$$

rendering gradient-based optimisation exponentially difficult to initialise for large n. This result extends beyond statistical fluctuations: by symmetry, the expected gradient is also zero at random initialisation, so individual gradient measurements provide no useful directional information.

Several mitigating strategies have been proposed. Local cost functions, where the observable is restricted to a constant-size subsystem, exhibit more favourable gradient scaling than global observables [47]. Problem-inspired ansätze encoding structural prior knowledge resist barren plateaus by restricting the circuit family to a low-dimensional subspace. Layerwise training [43] initialises circuit blocks incrementally to avoid the random landscape regime. However, no universal mitigation is currently available; deeper circuits with greater expressive power consistently exhibit worse gradient concentration.

Barren plateaus also arise in noise-dominated regimes [55]. Hardware noise effectively flattens the cost landscape independently of the optimisation method, compounding the initialisation difficulty in practice.

Generalisation. Beyond trainability, generalisation theory for QML models remains an active research area. Recent work [56] relates the effective dimension of quantum model families to generalisation bounds, suggesting that highly expressive

circuits may overfit small training datasets. Expressivity alone therefore does not guarantee practical advantage; inductive bias and noise robustness are at least as critical for performance on realistic data. Noise-aware training strategies and classical pre-processing pipelines have been proposed to improve stability under realistic hardware conditions [57].

2.3.3 Dequantization and the Limits of Quantum Kernel Advantage

A significant development in theoretical QML has been the formalisation of *dequantization*: the construction of classical algorithms that match the asymptotic performance of proposed quantum algorithms under comparable data-access assumptions. Tang [58] showed that the exponential speedup attributed to the quantum recommendation algorithm of Kerenidis and Prakash could be reproduced classically given sample-and-query access to the input matrix. Subsequent work extended analogous arguments to quantum principal component analysis and quantum-assisted linear systems [59].

These results do not exclude quantum advantage in general, but they impose a significant evidentiary burden on QML proposals that rely on amplitude-encoding of classically accessible data. If the claimed speedup depends primarily on efficient quantum data loading, a classical algorithm with equivalent access may reproduce it. The key implication is that proposals based on standard feature-encoding circuits must demonstrate advantage that is attributable to quantum processing rather than to input-access assumptions.

Quantum kernel expressivity versus classical surrogates. Quantum kernel methods evaluate $K(x,x') = |\langle\Phi(x)|\Phi(x')\rangle|^2$ using hardware-implemented feature maps. Huang et al. [51] established conditions under which quantum kernels can outperform classical kernels: specifically, when the dataset has structure that is efficiently expressible by the quantum feature map but computationally costly to capture with classical polynomial-complexity kernels. However, this advantage is dataset-dependent. For generic datasets, quantum kernel matrices can become exponentially concentrated, effectively becoming uninformative, in a manner analogous to the barren plateau in variational models [60].

Classical surrogate models based on random Fourier feature approximations and classical kernel methods have been shown to match quantum kernel performance on standard classification benchmarks [51]. This indicates that quantum kernel advantage, when present, is likely narrow, instance-specific, and dependent on a deliberate alignment between the hardware feature map and the data geometry, rather than a broadly applicable property.

Implications for the present thesis. These theoretical constraints motivate the specific algorithmic designs adopted in this work. Quantum Reservoir Computing (Chapter 3) avoids variational training entirely: the quantum component acts as a fixed nonlinear projection, and only a classical linear readout layer is optimised. This design sidesteps the barren plateau problem and reduces the number of circuit evaluations required for training. The Quantum Evolution Kernel (Chapter 3) does not rely on classical feature-encoding circuits; instead, it exploits hardware-native Rydberg dynamics to construct a kernel that is computationally costly to simulate classically but directly executable on neutral-atom hardware. Both methods therefore occupy a regime in which hardware-native properties, rather than data-encoding capacity, drive the potential for practical utility.

2.3.4 Quantum Reservoir Computing

Quantum Reservoir Computing (QRC) has gained renewed interest as a NISQ-friendly approach [61]. Recent experimental realizations demonstrate robustness to noise and reduced training overhead [62].

Since QRC trains only classical readout weights, it minimizes quantum circuit reconfiguration and may better align with batch-execution constraints.

2.3.5 Scalability Challenges

Practical QML deployments face a combination of obstacles: limited qubit counts restrict model size, kernel matrix estimation is expensive, training requires iterative circuit evaluations, and every hardware invocation incurs access latency. Infrastructure-level optimization therefore becomes as important as algorithmic design, particularly in hybrid QC–HPC systems.

2.4 Quantum Algorithms for Combinatorial Optimization

Combinatorial optimization represents one of the most studied application domains for near-term quantum computing. Many NP-hard problems can be reformulated as Quadratic Unconstrained Binary Optimization (QUBO) or Ising Hamiltonians [63]. This mapping enables the use of gate-based variational algorithms as well as quantum annealing approaches.

2.4.1 QUBO and Ising Formulations

A general QUBO problem can be expressed as:

$$\min_{x \in \{0,1\}^n} x^T Q x,$$

which can be equivalently mapped to an Ising Hamiltonian:

$$H = \sum_i h_i Z_i + \sum_{i<j} J_{ij} Z_i Z_j.$$

Such formulations encompass a broad range of problems, including Max-Cut, graph coloring, the Traveling Salesman Problem, portfolio optimization, and scheduling problems. The mapping overhead (e.g., penalty encoding for constraints) often increases qubit requirements and interaction density.

2.4.2 Quantum Approximate Optimization Algorithm

QAOA [64] is the most prominent gate-based approach for combinatorial optimization in the NISQ era.

Research on QAOA has since explored parameter concentration phenomena [65], performance scaling with depth p [66], analytical parameter setting strategies [67], warm-start variants [68], and noise-resilient formulations [69].

While low-depth QAOA can outperform classical heuristics on specific structured instances, large-scale advantage remains unproven.

Moreover, QAOA requires repeated expectation value estimation, making it structurally similar to VQE from an execution standpoint.

2.4.3 Quantum Annealing and Hybrid Solvers

Quantum annealers (e.g., D-Wave systems) implement transverse-field Ising dynamics. Hybrid classical-quantum decomposition strategies have gained traction in recent years [70]. These approaches partition large problems and solve subproblems using quantum resources.

However, embedding overhead and limited connectivity remain significant constraints.

2.4.4 Decomposition and Hybrid Cooperation

Large-scale optimization instances often exceed available qubit counts. Hybrid decomposition strategies therefore split the problem into a classical master problem and a set of quantum subproblem evaluations. Recent works explore Benders-like decomposition for QUBO [71], large-neighborhood search with quantum subroutines [72], and iterative classical-quantum refinement [69].

These hybrid approaches strongly depend on execution latency and resource availability, making them sensitive to infrastructure design.

2.4.5 Complexity and Practical Limitations

From a theoretical perspective, rigorous quantum advantage for generic NP-hard problems remains unlikely in worst-case settings. Recent analyses emphasize that any advantage will likely be instance-dependent and heuristic [73].

In practice, current limitations include limited circuit depth, sampling noise, high shot requirements, and queue delays on shared quantum hardware. Combinatorial optimization therefore provides an ideal testbed for studying platform-aware hybrid

execution, particularly when large parameter sweeps or decomposition workflows are required.

2.4.6 Comparative Analysis of QML and Optimization Workloads

Before moving to the next Section, it is useful to provide a comparative overview of the main hybrid algorithms for the two considered application domains. Table 2.1 shows the main characteristics of both in terms of goal, required quantum resources and limitations.

Table 2.1 Comparison of workload characteristics for QML and combinatorial optimization on NISQ devices.

Aspect	**Variational QML**	**Quantum Kernels**	**Optimization (QAOA / Hybrid)**
Primary Goal	Classification / Regression	Feature-space embedding	Energy minimization
Quantum Depth	Shallow–Moderate	Typically shallow	Depth increases with p
Iterations Required	High (training loops)	$O(N^2)$ kernel evaluations	High (parameter sweeps)
Shot Requirements	High (gradients)	Very high (pairwise overlaps)	High (expectation estimation)
Sensitivity to Noise	High (gradients unstable)	Moderate	High (objective bias)
Barren Plateau Risk	Present	Absent	Present (deep QAOA)
Classical-Quantum Coupling	Tight (per batch)	Loose but frequent	Tight iterative loop
Scalability Bottleneck	Gradient variance	Kernel matrix scaling	Depth + shot cost
Infrastructure Sensitivity	Very high	High	Very high

2.5 High-Performance Computing Systems and Batch Scheduling

High-Performance Computing (HPC) systems are traditionally designed for large-scale, long-running, homogeneous workloads such as numerical simulations, climate modelling, and large-scale data analytics. Resource management is handled by batch schedulers such as SLURM, PBS, or LSF, which allocate compute nodes to submitted jobs based on priority queues, backfill policies, and fairness constraints.

2.5.1 Batch Scheduling and Malleable Workloads

SLURM, one of the most widely adopted open-source schedulers, supports priority queues, backfilling, reservations, and resource-aware job placement [74]. Its scheduling model assumes that workloads declare resource requirements at submission time and hold them for the duration of the job. Jobs are expected to reserve resources for relatively long durations, to execute with predictable and stable scaling behaviour, and to use homogeneous resource types such as CPU cores, GPU cards, or memory banks.

Research on malleable and elastic scheduling [75, 76] has extended this model to accommodate workloads whose resource footprint can vary during execution. Malleability allows a running job to release or acquire compute nodes in response to system load, improving overall utilisation. These mechanisms have been demonstrated for iterative numerical solvers and simulation pipelines. However, they remain designed for workloads whose execution phases are locally controlled: both the job and the resources it uses reside within the same administrative and physical domain.

2.5.2 Heterogeneous Accelerator Integration: GPU and FPGA Co-Scheduling

The integration of GPU and FPGA accelerators into HPC environments is well-studied, and scheduling frameworks for these devices have matured within the existing batch infrastructure [77]. Several properties of these accelerators make scheduling tractable without fundamental changes to the batch model.

GPUs are physically co-located with compute nodes, connected via PCIe or NVLink. Host-to-device data transfers are deterministic and bounded, typically completing in microseconds. SLURM supports GPU allocation natively through Generic Resource (GRES) tokens: a job declares its GPU requirement at submission, and the scheduler guarantees exclusive access on the allocated nodes. GPU driver and firmware layers handle thermal management, frequency scaling, and error detection transparently, without any involvement from the job scheduler. Execution is synchronous within the job allocation: CPU and GPU computation phases interleave predictably, and total runtime can be estimated from benchmark profiles collected on the same hardware.

FPGA integration is less standardised but structurally similar [78]. Bitstream configuration and partial reconfiguration introduce setup overhead, but FPGA accelerators remain physically local to the host node, communicate via deterministic PCIe paths, and execute under direct software control without external dependencies.

A key shared property of GPU and FPGA scheduling is that accelerator performance is stationary over the allocation window. A GPU's throughput and error characteristics do not vary as a function of external calibration cycles, usage demand from other organisations, or time-of-day effects. The scheduler allocates a well-characterised resource, tracks it as occupied, and recovers it upon job completion. This stationarity makes resource accounting straightforward and allows job runtime estimates to be meaningful.

2.5.3 The Quantum Integration Challenge: Novel Scheduling Constraints

Quantum Processing Units (QPUs) introduce a qualitatively different set of constraints that have no direct analogue in GPU or FPGA integration. Under current access models, QPUs are operated by cloud providers and accessed via REST APIs from platforms such as Amazon Braket or IBM Quantum. The HPC centre neither owns nor controls the quantum hardware, and its performance characteristics are not observable in real time.

Five properties of QPU access models diverge structurally from classical accelerator scheduling. The first is non-deterministic queue delay: jobs submitted to cloud QPU queues experience wait times that depend on global demand, maintenance

schedules, and calibration cycles. Delays of minutes to hours are common and cannot be bounded at submission time, so an HPC job holding allocated classical nodes while waiting for QPU results incurs idle classical resource cost. The second is calibration-dependent performance: QPU gate fidelities drift between calibration runs, which typically occur on a daily cycle. Device performance at the time of execution is not under the control of the HPC scheduler, and experimental campaigns must account for this variability, which introduces a time-dependent noise profile with no GPU analogue.

The third property is metered access: cloud QPU usage is subject to per-job or per-period shot budgets, introducing a form of resource scarcity metered independently of classical compute time. Variable shot counts across variational iterations further complicate resource accounting. The fourth is asynchronous result collection: unlike GPU execution, which completes synchronously within the host job, QPU results must be polled or retrieved after submission. The classical workflow must therefore manage asynchronous synchronisation and re-assembly of results across iterations, a control pattern that standard batch schedulers do not currently support.

The fifth property is non-exclusive and shared access: QPU time is shared globally across all users of the cloud platform. No equivalent to a dedicated-node allocation exists for most QPU providers, and two sequential circuit evaluations from the same job may execute on hardware in different calibration states.

Some providers have begun exposing QPU access through Kubernetes-based cloud-native interfaces (e.g., Qiskit Runtime primitives [79]), which improve portability and abstraction but do not resolve the underlying non-determinism, calibration drift, and shared-access characteristics described above.

These properties collectively define a scheduling problem that crosses organisational and geographical boundaries, involves asynchronous and non-deterministic external dependencies, and operates under resource constraints with no direct precedent in classical HPC workflows. The standard batch scheduler abstraction, allocate, execute, release, does not map onto the QPU access model without significant adaptation.

These mismatches motivate the virtualization layers and co-scheduling mechanisms developed in Chapter 5, where a vQPU abstraction is introduced to expose QPU resources within a SLURM-compatible scheduling framework.

2.6 Challenges in Quantum-HPC Integration

The integration of quantum processors into HPC environments introduces novel systems-level challenges that extend beyond algorithmic considerations.

2.6.1 Latency Amplification

Variational algorithms may require thousands of circuit evaluations per optimization run. If each invocation experiences network latency or queue delay, total execution time may become dominated by communication overhead rather than quantum computation itself.

Recent architectural studies [80, 81] emphasize that quantum acceleration cannot be evaluated independently from orchestration cost.

2.6.2 Resource Fragmentation and Idle Time

Hybrid workloads often involve alternating classical and quantum phases. Without coordinated scheduling, classical nodes may idle while waiting for quantum results, reducing overall system utilization.

Emerging work on quantum-aware resource management [82] proposes tighter coupling between classical and quantum scheduling domains.

2.6.3 Co-Scheduling and Reservation Models

Traditional HPC schedulers do not natively support joint reservation of remote quantum devices. This leads to ad-hoc synchronization strategies or over-provisioning of classical resources.

Research on co-scheduling accelerators [83] provides useful conceptual foundations, but quantum devices introduce additional constraints such as calibration windows, limited shot quotas, and remote API access models.

2.6.4 Virtualization and Abstraction Layers

Recent proposals advocate for virtualization layers that abstract quantum devices as schedulable resources within HPC infrastructures [84].

Such approaches aim to expose quantum resources via standardized interfaces, to enable unified resource accounting, to support dynamic reservation mechanisms, and to improve reproducibility and performance predictability.

These infrastructural challenges motivate the virtualization and platform-aware resource management strategies developed in Chapter 5.

2.7 Critical Assessment and Research Gaps

Despite rapid progress in both quantum hardware and hybrid algorithms in recent years, several structural limitations remain unresolved. A critical examination of the state of the art reveals a number of gaps that motivate the contributions of this dissertation.

2.7.1 Algorithm-Hardware Mismatch

Most variational and hybrid algorithms are evaluated assuming idealized access to quantum processors. Queueing delays on shared quantum backends, limited shot budgets, the latency between classical optimization and quantum evaluation, and device calibration drift are often neglected in such evaluations.

However, practical deployments reveal that execution overhead frequently dominates algorithmic runtime. Thus, performance cannot be assessed solely at the circuit level.

2.7.2 Underexplored Execution Models

The majority of QML and QAOA studies focus on algorithmic properties such as expressibility, convergence, and approximation ratios. Relatively little attention has been paid to the cost of iterative invocation in distributed systems, to resource contention in HPC environments, to the patterns of interaction between hybrid

applications and batch schedulers, and to the mismatch between reservation-based allocation and phase-alternating workloads.

As shown in recent system-level studies [49], execution architecture can significantly alter effective scalability.

2.7.3 Scalability Bottlenecks Beyond Qubits

While qubit count often dominates public discourse, practical scalability is constrained by shot complexity, gradient variance, the growth of kernel matrices with dataset size, and the dimensionality of parameter sweeps.

These bottlenecks are computational and infrastructural rather than purely quantum.

2.7.4 Lack of Integrated QC–HPC Abstractions

Current quantum cloud platforms provide job-based interfaces that do not seamlessly integrate with traditional HPC schedulers such as SLURM. Hybrid applications therefore operate across heterogeneous control planes, which leads to synchronization inefficiencies, idle classical resources, reduced throughput, and limited reproducibility.

This fragmentation motivates virtualization layers and execution abstractions capable of bridging quantum and HPC environments.

2.7.5 Research Questions Emerging from the Gap Analysis

The gaps identified above map directly onto the research questions formulated in Section 1.3. The algorithm–hardware mismatch raises the question of how hybrid algorithms can be co-designed with the infrastructure that executes them; this is the subject of RQ1, addressed in Chapters 3 and 4. The lack of workload characterization motivates RQ2, the search for execution models that capture the iterative and latency-sensitive behavior of hybrid applications. The absence of integrated QC–HPC abstractions leads to RQ3, which asks how quantum resources can be exposed within HPC scheduling frameworks; the vQPU abstraction developed in Chapter 5 responds

to this gap. Finally, the reservation and utilization inefficiencies observed across the reviewed literature motivate RQ4, on resource management strategies, including malleability, that make iterative quantum workloads resource-aware. Together, these questions define the architectural and system-level focus of the remainder of this thesis.

Chapter 3

Quantum Machine Learning on NISQ Devices

This Chapter examines Quantum Machine Learning as one of the most prominent application domains for hybrid quantum–classical computing in the NISQ era. Rather than treating QML as a uniform class of methods, the Chapter analyzes different paradigms that exhibit distinct algorithmic structures, training requirements, and infrastructure-level implications. After introducing the general characteristics of QML under realistic hardware constraints, the discussion focuses on three representative directions: variational quantum models, Quantum Reservoir Computing, and quantum kernel methods. The Chapter then grounds this analysis through concrete case studies, which are used not only to illustrate algorithmic design choices, but also to highlight how different QML approaches interact with limitations in qubit count, noise, data encoding, and execution models. This perspective provides the basis for the comparative and infrastructure-aware discussion developed at the end of the Chapter.

3.1 Quantum Machine Learning in the NISQ Regime

Quantum Machine Learning has emerged as one of the most actively investigated application domains in the Noisy Intermediate-Scale Quantum era. By combining concepts from quantum computing and classical machine learning, QML aims at leveraging quantum mechanical effects—such as superposition, entanglement, and

interference—to enhance learning models or to provide computational advantages for specific tasks.

In the current technological landscape, however, QML must operate under stringent hardware constraints: limited qubit counts, restricted connectivity, short coherence times, and non-negligible noise levels. As a consequence, most practical approaches rely on hybrid quantum-classical paradigms, where parameterized quantum circuits are embedded within classical optimization loops.

3.1.1 Conceptual Foundations

Quantum Machine Learning encompasses a broad class of algorithms that combine quantum computation with statistical learning theory. A widely adopted taxonomy distinguishes between:

- **Quantum-enhanced classical models**, where quantum routines accelerate subroutines of classical algorithms;
- **Variational hybrid models**, where a parameterized quantum circuit is trained through classical optimization;
- **Fully quantum models**, typically requiring fault-tolerant quantum computation.

While early theoretical proposals—such as the Harrow–Hassidim–Lloyd (HHL) algorithm for linear systems [85]—suggested exponential speedups, their practical implementation requires deep circuits and quantum error correction, which are beyond current hardware capabilities.

Consequently, the NISQ era has shifted attention toward hybrid variational methods [41], which trade asymptotic speedups for hardware feasibility. In these models, a quantum processor evaluates expectation values of parameterized circuits, while a classical optimizer updates the parameters based on a cost function.

3.1.2 Data Encoding Strategies

A central challenge in QML concerns the embedding of classical data into quantum states. This step, known as *quantum feature mapping*, directly affects expressivity, circuit depth, and hardware feasibility.

Common encoding strategies include:

- **Basis encoding**, where classical bits are mapped to computational basis states;
- **Amplitude encoding**, where data are embedded into quantum amplitudes;
- **Angle encoding**, where classical features modulate rotation gates;
- **Hamiltonian encoding**, where data define the parameters of time-evolution operators.

Amplitude encoding offers exponential compression in principle, but requires state preparation circuits whose depth scales unfavorably with input dimension. In contrast, angle encoding produces shallow circuits and is therefore widely adopted in NISQ-compatible architectures.

From a cross-layer perspective, encoding cost is not merely algorithmic: it directly impacts compilation time, transpilation overhead, and total quantum invocation count. In hybrid HPC–QC systems, repeated state preparation inside iterative loops can dominate overall runtime.

3.1.3 Trainability and Barren Plateaus

An important theoretical challenge in variational QML concerns the so-called *barren plateau* phenomenon [46], where the gradient of the cost function vanishes exponentially with system size for randomly initialized circuits.

This effect limits scalability and highlights the importance of:

- Problem-inspired ansätze;
- Hardware-efficient circuit design;
- Layer-wise training strategies;

- Careful parameter initialization.

These considerations reinforce the central thesis of this work: algorithmic performance cannot be decoupled from hardware characteristics. Expressivity, trainability, and noise resilience are tightly coupled to device topology and calibration quality.

3.1.4 Computational Model of Hybrid QML

In practical NISQ implementations, a QML training loop typically follows this structure:

1. Classical preprocessing of input data;
2. Quantum state preparation and parameterized circuit execution;
3. Measurement and expectation value estimation (with multiple shots);
4. Classical parameter update (e.g., gradient descent, SPSA);
5. Iteration until convergence.

The quantum processor acts as a co-processor for evaluating nonlinear feature maps or parametrized transformations. However, the dominant cost often arises from:

- Repeated circuit executions;
- Shot statistics;
- Queueing and latency overhead;
- Classical-quantum synchronization.

Therefore, evaluating QML performance requires considering not only accuracy metrics but also execution models and infrastructure integration, which are addressed in later Chapters.

3.2 Variational Quantum Models and Quantum Neural Networks

This Section discusses variational quantum models, which represent one of the most widely studied approaches to Quantum Machine Learning in the NISQ regime. These methods rely on parameterized quantum circuits embedded within classical optimization loops, and therefore exemplify the tightest form of hybrid interaction between quantum and classical computation. The Section introduces their main architectural elements, reviews their relation to the broader notion of quantum neural networks, and examines the main training strategies adopted in practice. Particular attention is devoted to the trade-off between expressive power and hardware feasibility, since this balance is one of the central issues that determines the applicability of variational QML on present-day devices.

3.2.1 Parameterized Quantum Circuits

Variational Quantum Circuits (VQCs), also referred to as parameterized quantum circuits (PQCs), represent the dominant paradigm for NISQ QML [86]. A VQC consists of alternating layers of:

- Parameterized single-qubit rotations;
- Entangling gates compatible with hardware connectivity.

Formally, a VQC implements a unitary transformation:

$$U(\theta) = \prod_{l=1}^{L} U_l(\theta_l),$$

where θ denotes the trainable parameters and L the number of layers.

The cost function is typically defined as:

$$C(\theta) = \langle 0|U^{\dagger}(\theta)\hat{O}U(\theta)|0\rangle,$$

with $\hat{O}$ an observable encoding the learning objective.

3.2.2 Quantum Neural Networks

Quantum Neural Networks (QNNs) generalize VQCs by drawing an analogy with classical neural architectures [87]. Layers of parameterized gates mimic affine transformations, while measurement outcomes play the role of activation functions.

Different architectural families have been proposed:

- **Hardware-efficient ansätze**, designed to minimize circuit depth;
- **Problem-inspired ansätze**, incorporating domain knowledge;
- **Data re-uploading models**, where classical data are repeatedly injected across layers [88].

While QNNs provide a flexible modeling framework, their practical effectiveness depends strongly on noise levels and sampling cost. Unlike classical neural networks, gradients must be estimated through repeated circuit evaluations, commonly using the parameter-shift rule.

3.2.3 Training Strategies

Hybrid training can be performed using:

- Gradient-based optimizers (e.g., Adam, L-BFGS);
- Gradient-free methods (e.g., COBYLA, SPSA);
- Evolutionary strategies.

In NISQ settings, stochastic and noise-resilient optimizers are often preferred. The Simultaneous Perturbation Stochastic Approximation (SPSA) method has proven particularly effective under shot noise.

From an infrastructure standpoint, the choice of optimizer directly impacts:

- Number of quantum circuit evaluations per iteration;
- Parallelizability of gradient computation;

- Communication overhead in distributed environments.

This is especially relevant when QML workloads are deployed in HPC–QC integrated systems, where each quantum call may involve remote submission, queueing, and result retrieval.

3.2.4 Expressivity vs. Hardware Constraints

A central trade-off in VQC design concerns expressivity versus implementability. Deep circuits may approximate complex decision boundaries but exceed coherence limits. Shallow circuits preserve fidelity but may lack representational power.

Hardware topology further constrains entangling patterns. On superconducting architectures with nearest-neighbor connectivity, additional SWAP gates may significantly increase depth after transpilation.

Therefore, the design of QML models must be platform-aware. Later Chapters demonstrate how these algorithmic considerations propagate upward to execution models and resource management strategies in hybrid HPC–QC infrastructures.

3.3 Quantum Reservoir Computing

This Section focuses on Quantum Reservoir Computing, a QML paradigm that is particularly appealing in the NISQ setting because it shifts the training burden away from the quantum device. Instead of optimizing all quantum parameters iteratively, QRC exploits the dynamics of a fixed quantum system to generate nonlinear representations that are then processed by a classical readout layer. After recalling the conceptual relation between classical and quantum reservoirs, the Section introduces the dynamical model underlying QRC and discusses how this paradigm can be adapted to current hardware constraints. The Section concludes with the first QML case study of the thesis, which uses QRC to address a classification task and serves as a concrete example of a hybrid workflow in which the quantum processor acts primarily as a feature generator.

3.3.1 From Classical to Quantum Reservoirs

Reservoir Computing (RC) is a computational framework originally developed for processing temporal data through recurrent dynamical systems [89, 90]. In classical RC, a high-dimensional dynamical system—called the *reservoir*—is driven by an input signal, while only a linear readout layer is trained. The internal dynamics remain fixed.

The key idea is to exploit the intrinsic nonlinear dynamics of the reservoir to project input data into a rich feature space where linear separation becomes possible. Two central properties characterize an effective reservoir:

- **Separation property**: different inputs lead to distinguishable reservoir states;
- **Fading memory property**: the system retains information about past inputs while gradually forgetting distant history.

QRC extends this paradigm by using a quantum dynamical system as the reservoir [91]. Instead of classical recurrent connections, the quantum evolution of a many-body system generates a high-dimensional Hilbert-space embedding of the input signal.

Unlike variational quantum circuits, QRC does not require training of quantum parameters. The only trainable component is the classical readout layer. This significantly reduces the optimization overhead and mitigates barren plateau issues discussed in 3.1.

3.3.2 Quantum Dynamical Model

In a typical QRC setup, the reservoir consists of N interacting qubits described by a Hamiltonian H. The input signal $u(t)$ modulates either:

- The initial state preparation,
- A subset of control parameters in H,
- Or a sequence of unitary operators applied at discrete time steps.

The reservoir state evolves according to:

$$\rho(t+1) = U(u(t))\rho(t)U^{\dagger}(u(t)),$$

where $U(u(t))$ is an input-dependent unitary transformation. Observable expectation values:

$$x_i(t) = \mathrm{Tr}[\rho(t)O_i],$$

are sampled and fed into a classical linear regressor:

$$y(t) = \sum_i w_i x_i(t).$$

The richness of the quantum feature space arises from:

- Exponential Hilbert space dimension (2^N),
- Nonlinear dependence of observables on the input history,
- Many-body entanglement dynamics.

Importantly, quantum measurements introduce stochasticity and partial state collapse, which must be carefully managed in practical implementations.

3.3.3 Large-Scale Quantum Reservoir Learning with an Analog Quantum Computer

In [92], the reservoir is realized through continuous-time evolution of an analog programmable quantum simulator. Instead of compiling digital gate sequences, the approach exploits intrinsic many-body dynamics governed by a tunable Hamiltonian.

Analog Reservoir Dynamics

The system evolves under a transverse-field Ising Hamiltonian

$$\hat{H}(t) = \sum_{i<j} J_{ij}\hat{\sigma}_i^z\hat{\sigma}_j^z + \sum_i h_i(t)\hat{\sigma}_i^x, \tag{3.1}$$

where couplings J_{ij} define fixed interactions and input signals modulate local transverse fields via

$$h_i(t) = h_i^{(0)} + \alpha_i u(t). \tag{3.2}$$

The state evolution

$$|\psi(t+\Delta t)\rangle = e^{-i\hat{H}(t)\Delta t}|\psi(t)\rangle \tag{3.3}$$

naturally generates nonlinear mixing and entanglement. No quantum parameters are trained; only the classical readout weights are optimized.

Feature Extraction and Learning

Reservoir features consist of expectation values of few-body observables,

$$x_k(t) = \langle\psi(t)|\hat{O}_k|\psi(t)\rangle, \tag{3.4}$$

which implicitly encode high-order correlations induced by the interacting dynamics. A linear regressor produces the output signal.

Empirically, memory capacity and nonlinear task performance increase with reservoir size. Because evolution is analog and continuous, scaling the number of qubits does not require increasing circuit depth, distinguishing this approach from digital implementations.

Noise and Execution Characteristics

Performance degrades gradually under realistic decoherence, reflecting the absence of delicate variational interference patterns. The analog implementation shifts complexity from circuit synthesis to control calibration, while maintaining shallow effective depth.

From a computational perspective, execution involves repeated measurement sampling combined with classical regression training. The absence of iterative quantum-classical feedback simplifies scheduling and enables streaming data injection, making analog QRC well suited to large-scale hybrid infrastructures.

We now proceed to define the methodology of a case study which is part of the original work of the author of this thesis. The objective of such is to provide an example of practical implementation of QRC methodologies to characterize the hybrid workflow. This will allow us to identify and evaluate bottlenecks and limitations of the state-of-the-art at multiple levels of the stack.

3.3.4 Case Study I: Classification with QRC

The case study investigates the application of QRC to a real-world binary classification problem in the financial domain. The reservoir is used as a nonlinear feature transformation mechanism for structured tabular data.

The primary objective of this Section is methodological: to define and analyze a full hybrid quantum–classical pipeline suitable for execution on NISQ-era analog hardware, and to assess its algorithmic structure independently from the final implementation and performance, which are discussed in Chapter 6. It is also important to note that, while the specific application is completely original, the framework closely follows the one described in Section 3.3.3. Finally, we want to remark that the work is under review process for publication at the time of writing, but a preprint is already available [1].

Problem Definition and Dataset

The selected benchmark concerns credit card default prediction. Given a set of attributes describing customers' financial behavior and demographic information, the task is to predict whether a client will default on their payment in the following month. The problem is formulated as a supervised binary classification task:

$$f : \mathbb{R}^d \to \{0, 1\}, \tag{3.5}$$

where d denotes the number of input features and the target variable encodes default (1) or non-default (0).

We chose a well-know public dataset [93] consisting of 30,000 samples and 23 explanatory variables, including:

- demographic features (e.g., gender, education level, marital status, age),
- credit limit,
- repayment status over the previous six months,
- bill statement amounts,
- previous payment amounts.

The dataset is intrinsically imbalanced: approximately 22% of the samples correspond to default cases, while 78% correspond to non-default instances. This imbalance strongly affects model training and evaluation, requiring dedicated resampling and metric-selection strategies.

Data Preprocessing

Preprocessing plays a critical role, both for statistical robustness and for compatibility with quantum hardware constraints. The preprocessing pipeline is structured as follows.

Data cleaning and consistency checks. Missing values are not present in the dataset; however, categorical features encoded as integers are explicitly treated as nominal variables where appropriate (e.g., education level). Outliers in continuous variables (such as bill amounts) are retained but monitored through robust scaling techniques to avoid undue influence on normalization.

Feature encoding. Categorical features are processed through either:

- one-hot encoding, or
- ordinal encoding when natural ordering exists.

The choice depends on dimensionality constraints and its impact on the reservoir input size. Since quantum encoding resources are finite, dimensionality reduction considerations are incorporated early in the pipeline.

Feature scaling. All continuous variables are normalized to a bounded interval. Two alternative scaling strategies are considered:

- Standardization (zero mean, unit variance),
- Min–max scaling to a predefined interval $[a, b]$.

The second approach is particularly relevant when mapping classical features to Hamiltonian parameters (e.g., detuning amplitudes), which must remain within hardware-allowed ranges. For instance, when using local detuning encoding, input features are linearly mapped to:

$$\Delta_i \in [-\Delta_{\max}, \Delta_{\max}], \tag{3.6}$$

where $\Delta_{\max}$ is determined by the hardware.

Dimensionality reduction (optional). To control the number of quantum control parameters and mitigate noise amplification, Principal Component Analysis (PCA) is optionally applied. The reduced feature space preserves a predefined fraction (e.g., 95%) of total variance. This step is particularly useful when comparing position encoding (which depends on atom arrangement) to detuning encoding (which scales with qubit count).

Handling Class Imbalance

Given the imbalanced nature of the problem, several resampling strategies are evaluated during training:

- *Synthetic Minority Over-sampling Technique (SMOTE)* and *K-Means SMOTE (K-SMOTE)* [94] for oversampling,
- *Cluster Centroids (CC)* [95] for undersampling,

- A baseline case with no resampling, allowing the models to handle the imbalance inherently.

Resampling is performed exclusively on the training set to avoid information leakage. Stratified splitting ensures consistent class proportions between training and test sets. In all experiments, evaluation is conducted on untouched test data.

Hybrid Quantum–Classical Pipeline

The overall architecture follows the reservoir computing paradigm:

1. Classical preprocessing,
2. Quantum reservoir transformation,
3. Classical readout training.

Figure 3.1 provides a schematic overview of the hybrid pipeline.

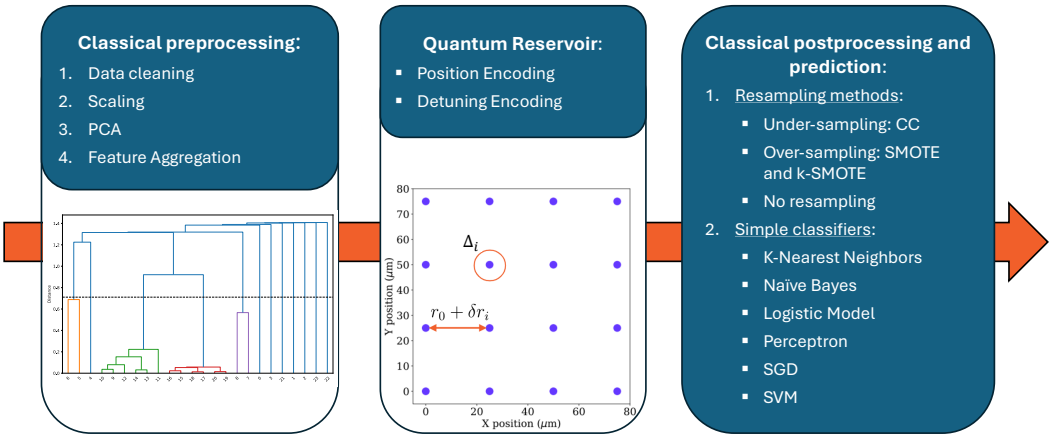


Fig. 3.1 Detailed hybrid pipeline for classification with QRC, including preprocessing, resampling, quantum encoding, time evolution, measurement, and classical classification.

Quantum Reservoir Implementation

Even if the the objective of this Chapter is to characterize prototypical hybrid algorithms in terms of workload independently of the actual target quantum platform, we think it is still useful to introduce the basic concepts of the chosen hardware, since it can help the reader to better understand the overall methodology, without loss of generality. Indeed, QRC routines can be implemented on digital quantum computers, for example using the scheme proposed in [96], which makes use of mid-circuit measurements and reset. However, in this case we decided to target a 256-qubit analog neutral atom simulator, namely QuEra's Aquila [97], as it provides large number of qubits and high degrees of programmability, flexibility and scalability. Therefore, the reservoir is implemented on quantum platform governed by the Hamiltonian:

$$\hat{H}(t) = \frac{\hbar\Omega(t)}{2} \sum_{i=1}^{n} \left(e^{i\phi(t)} |0_i\rangle \langle 1_i| + e^{-i\phi(t)} |1_i\rangle \langle 0_i| \right) - \sum_{i=1}^{n} \hbar\Delta_i(t)\hat{n}_i + \sum_{j<i} V_{ij}\hat{n}_i\hat{n}_j, \tag{3.7}$$

where $\Omega(t)$ is the time-dependent global Rabi drive, $\phi(t)$ its relative phase, $\Delta_i(t)$ the site-dependent (or local) detuning, and $\hat{n}_i$ is the operator $|1_i\rangle \langle 1_i|$ that counts the atoms in the excited state and C_6 is the Rydberg interaction coefficient, whose value depends on the chosen Rydberg level. The last term, the Van der Waals interaction, is often rewritten using $V_{ij} = \frac{C_6}{r_{i,j}^6}$, where $r_{i,j}$ is the relative distance between the i^{th} and j^{th} atoms.

Given this Hamiltonian, two encoding schemes are investigated:

Local detuning encoding. Each feature component modulates the site-dependent detuning $\Delta_i(t)$. The mapping is linear:

$$\Delta_i(t) = \alpha x_i + \beta, \tag{3.8}$$

with (α, β) determined by scaling constraints.

Position encoding. Input features define relative inter-atomic distances r_{ij}, thereby modifying interaction strengths $V_{ij} \propto 1/r_{ij}^6$. This encoding exploits the geometric programmability of the neutral-atom array.

Temporal multiplexing. To increase effective feature dimensionality, observables are measured at multiple time steps:

$$\mathbf{z}(x) = \big(\langle \hat{O}_1(t_1)\rangle, \ldots, \langle \hat{O}_n(t_T)\rangle\big), \tag{3.9}$$

where T denotes the number of measurement times. This strategy increases expressive power without increasing qubit count.

Classical Readout Models

Consistent with the reservoir computing philosophy, only the readout layer is trained. The following classifiers are considered:

- Logistic Regression (with L_2 regularization),
- Support Vector Machines (linear and RBF kernels),
- Random Forest,
- Gradient Boosting,
- k-Nearest Neighbors,
- Multi-Layer Perceptron (shallow architecture).

Hyperparameters are optimized via cross-validation on the training set. For comparison, a fully classical Deep Neural Network (DNN) is trained directly on preprocessed features without the quantum reservoir transformation.

Evaluation Metrics

Given class imbalance, accuracy alone is insufficient. The following metrics are adopted:

- Precision,
- Recall,
- F1-score,
- Area Under the ROC Curve (AUC),
- Confusion matrix analysis.

In financial contexts, recall for the minority class (default detection) is particularly relevant, as false negatives correspond to underestimated credit risk.

Noise and Emulation

To isolate algorithmic capability from hardware limitations, three execution modes are compared:

1. Classical baseline (no quantum layer),
2. Noiseless classical emulation of the quantum dynamics,
3. Execution on physical quantum hardware.

The emulation allows us to probe the theoretical representational capacity of the QRC architecture. Hardware execution reveals sensitivity to decoherence, finite sampling noise, and control imperfections.

Infrastructure-Level Implications

This workload exhibits structural characteristics relevant to hybrid QC–HPC integration:

- repeated short quantum executions (one per sample or batch),
- high measurement shot counts,
- tight classical–quantum interaction loop.

Consequently, latency, queueing overhead, and resource allocation policies significantly influence end-to-end training time. These aspects directly connect to the virtualization and scheduling mechanisms discussed in Chapter 5.

The experimental evaluation of this case study is reported in Section 6.3.1. Classification accuracy—measured through F1-score and complementary metrics—is assessed across three execution modes: noiseless statevector emulation, shot-based emulation, and real-device execution on the Aquila 256-qubit neutral-atom platform. Hardware experiments use the `CARDS_1000` and `CARDS_2500` subsets, while the full `CARDS_30000` dataset is evaluated in emulation only, reflecting the practical constraints of remote QPU access with finite shot budgets. The central experimental questions are: whether the detuning-encoded reservoir achieves competitive classification performance relative to classical baselines (CRC and DNN) in the noiseless setting; whether hardware execution induces a significant or only partially recoverable degradation of predictive quality; and whether pipeline robustness improves as the downstream classifier is given access to more training samples.

3.4 Quantum Kernel Methods

This Section addresses quantum kernel methods, which provide a different route to QML by exploiting quantum evolution or quantum state overlaps to define similarity measures between data samples. In contrast with variational approaches, quantum kernels avoid the direct training of parameterized quantum models, but often shift the computational burden toward repeated evaluations of pairwise similarities. The Section first introduces the general idea of implicit feature maps in Hilbert space and the main overlap-estimation strategies used in practice. It then presents the Quantum Evolution Kernel considered in this thesis and discusses its use in graph learning. The Section concludes with a case study on graph classification, which illustrates both the potential of quantum kernels and the practical constraints that arise when embedding structured data into hardware-compatible quantum evolutions.

3.4.1 Implicit Feature Maps in Hilbert Space

Quantum kernel methods represent an alternative QML paradigm that avoids training parameterized quantum circuits. Instead, a quantum device is used to compute a kernel function:

$$K(x,x') = |\langle\phi(x)|\phi(x')\rangle|^2,$$

where $|\phi(x)\rangle = U(x)|0\rangle$ is a quantum feature map encoding input data.

Once the kernel matrix is computed, classical kernel methods—such as Support Vector Machines (SVMs)—are applied.

The theoretical appeal lies in the possibility that certain quantum feature maps generate classically intractable feature spaces [54]. However, practical advantage depends on noise resilience and efficient overlap estimation.

3.4.2 Overlap Estimation Techniques

Computing $K(x,x')$ requires estimating state overlaps. Common techniques include:

- Swap test circuits,
- Direct fidelity estimation,
- Projected measurement strategies.

In NISQ devices, shallow circuits and measurement-efficient protocols are essential to maintain fidelity. Moreover, the total number of quantum evaluations scales quadratically with dataset size, making kernel computation resource-intensive.

3.4.3 Quantum Evolution Kernel for Graph Learning

The Quantum Evolution Kernel (QEK) [98] implements a graph kernel by embedding graph topology directly into the Hamiltonian of a programmable quantum system. Instead of constructing a feature map through parametrized single-qubit rotations, the adjacency structure of a graph determines the interaction pattern of a many-body evolution.

Hamiltonian Encoding

Given a graph $G = (V,E)$ with $|V| = N$, each vertex is mapped to a qubit. The connectivity is encoded into an Ising-type Hamiltonian

$$\hat{H}_G = \sum_{(i,j)\in E} J_{ij}\hat{\sigma}_i^z\hat{\sigma}_j^z, \tag{3.10}$$

combined with a non-commuting mixing term

$$\hat{H}_\theta = \sum_{i\in V} \hat{\sigma}_i^y. \tag{3.11}$$

The evolution operator is constructed through alternating layers

$$U_G(\Lambda) = \prod_{\ell=0}^{p} e^{-it_\ell \hat{H}_G} e^{-i\theta_\ell \hat{H}_\theta}, \tag{3.12}$$

with $\Lambda = \{t_\ell, \theta_\ell\}$ classical hyperparameters. The initial state is typically $|+\rangle^{\otimes N}$ to ensure interference between computational basis components.

Distribution-Based Kernel Construction

After the Hamiltonian evolution, we sample the final state $|\psi_f\rangle$ measuring an observable $\hat{\mathcal{O}}$ with $\{|o_i\rangle, \ldots |o_K\rangle\}$ eigenstates. From that we build a probability distribution

$$P_G^{\hat{\mathcal{O}}}(\Lambda) = (p_1, \ldots p_K),\ p_i = |\langle o_i|\psi_f\rangle|^2$$

Similarity between two graphs G and G' is defined through the Jensen–Shannon divergence between their output distributions P_G, $P_{G'}$:

Given two probability distribution P and P', the Jensen–Shannon divergence is defined as:

$$\mathrm{JS}(P,P') = H\left(\frac{P+P'}{2}\right) - \frac{H(P)+H(P')}{2}$$

where $H(\cdot)$ is the Shannon entropy of a probability distribution, defined as

$$H(P) = -\sum_k p_k \log p_k$$

The kernel is then computed using:

$$K_\mu(G, G') = \exp\big(-\mu\, \mathrm{JS}(P_G, P_{G'})\big), \tag{3.13}$$

where μ is an hyperparameter that takes values in $\mathbb{R}$.

This construction guarantees a positive-definite kernel while capturing structural differences through coherent many-body dynamics. At depth $p = 1$, analytical expressions relate the kernel to correlation functions determined by the adjacency matrix, revealing connections with classical random-walk kernels but enriched by quantum interference effects.

Hyperparameters Λ are optimized classically, typically via Bayesian [99] optimization. The quantum processor is therefore used only for forward evaluations, without gradient estimation.

Hardware Realization and Scaling

The QEK is naturally suited to programmable neutral-atom arrays, where Rydberg interactions implement Ising couplings directly. Graph topology is thus embedded physically into qubit connectivity, minimizing digital circuit depth.

Kernel construction scales quadratically with dataset size due to pairwise similarity evaluation. However, each quantum execution remains shallow and hardware-native. Performance under moderate measurement noise remains stable, as the distribution-based similarity smooths statistical fluctuations.

From an execution standpoint, the QEK is characterized by shallow circuits, repeated sampling, and classical outer-loop hyperparameter optimization. These features make it compatible with batched quantum execution and asynchronous hybrid orchestration.

3.4.4 Case Study II: Graph Classification via Quantum Kernels

This section presents a large-scale experimental implementation of the QEK for graph classification on the Aquila neutral atom quantum simulator [97]. This work has been performed by the thesis' author and his collaborators not to introduce a new kernel formulation, but rather to demonstrate how such a kernel can be concretely implemented, optimized, and executed on real analog quantum hardware under realistic constraints.

The experiment leverages a 256-qubit neutral atom platform based on programmable arrays of ^{87}Rb atoms, where qubits are encoded in the ground and Rydberg electronic states and interactions arise from the Rydberg blockade mechanism. The implementation closely follows the QEK formalism introduced in Section 3.4.3. This work represents one of the first experimental hardware realizations of this type of quantum kernel, and has been published in [5].

Dataset and Problem Formulation The task consists of binary graph classification on the benchmark `PROTEINS` dataset [100, 101], one of the standard graph-learning datasets used in the literature. Each sample is a graph representing the structure of a protein, where nodes correspond to amino acids and edges connect amino acids within a prescribed spatial distance. Each graph carries a binary label indicating whether the protein is an enzyme or not. The full dataset contains 1,113 graphs (663 enzymes, 450 non-enzymes). Graph sizes range from 4 to 620 nodes (average 39.5) and edge counts from 5 to 1,049 (average 72.81). This variability immediately highlights the mismatch between the natural diversity of graph datasets and the strict geometric and qubit-budget constraints of current neutral-atom hardware. Complete dataset statistics and the node/edge distribution are reported in Appendix C.2.1 (Table C.4 and Figure C.3).

This dataset must be preprocessed to satisfy the hardware requirements of the target quantum platform. In the case of neutral-atom simulators, an embedding step reduces the number of graphs that can be used, as detailed in Section 6.1.3.

Quantum Evolution and Kernel Construction For each graph $G = (V, E)$ of the dataset, the QEK is constructed from the quantum evolution described in Section 3.4.3. Supposing two alternating layers of Ising-mixing Hamiltonians, the full

time evolution is parameterized by

$$\Lambda = \{\theta_0, t_0, \theta_1, t_1, \theta_2\}.$$

On neutral atom hardware, these layers correspond to global Rabi pulses interleaved with free evolution under the native Ising Hamiltonian, while in the case of universal digital quantum computers these terms can be implemented through parameterized (R_X/R_Y) and R_{ZZ} rotations, respectively.

After time evolution, the quantum state is measured in the computational basis. From the available bitstring measurements, we can classically compute the interaction energy observable as detailed in Section 6.2.3.

For each graph, repeated measurements produce an energy distribution P_G. To ensure comparability, distributions are discretized using fixed binning over a global energy interval. Then, we can use the kernel defined by Equation 3.13 to construct the kernel matrix. Finally, the latter is used to train a SVM classifier.

Hybrid Optimization of Pulse Parameters The kernel performance depends critically on Hamiltonian parameters Λ. In our experimental setup, direct optimization on hardware is impractical due to shot costs and queueing overhead. Therefore, a Bayesian optimization procedure was conducted using classical emulation (using the Julia language package Bloqade [102]) on a small subset of `PROTEINS`. The detailed description of this procedure and is reported in Section 6.1.3.

The parameter set was reformulated in terms of pulse durations:

$$\Lambda_{\text{BO}} = \{\tau_0, t_0, \tau_1, t_1, \tau_2\},$$

where the τ_i are the durations of the driven segments and satisfy $\theta_i = \Omega_0 \tau_i$, while the t_i represent the free evolution intervals.

Each optimization step required evaluating the full training pipeline, including kernel construction and 10-fold cross-validation of the SVM classifier. The objective function was the average F1-score.

The optimized parameters were subsequently deployed on a larger subset of the `PROTEINS` dataset executed on a 256-qubit neutral atom simulator.

Evaluation metrics Given the type of task (binary classification), we adopted similar metrics to the Credit Card Prediction described in Section 3.3.4:

- Accuracy
- Precision,
- Recall,
- F1-score

The classification performance of the QEK was compared with a classical kernel commonly used for graph-type data, i.e., the Shortest Path Kernel (SPK) [103].

Execution Workflow The complete hybrid workflow is summarized in Figure 3.2.

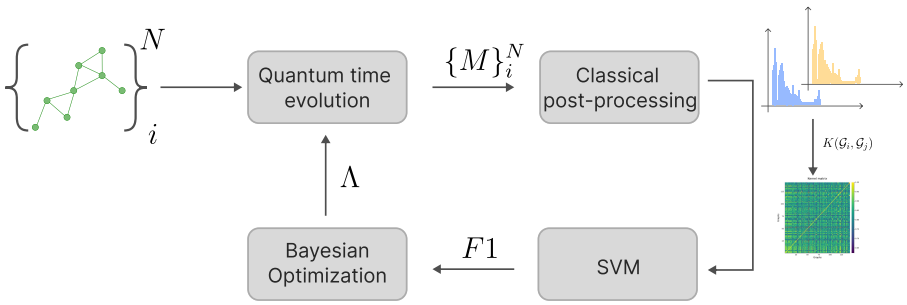


Fig. 3.2 Hybrid quantum-classical pipeline for graph classification via QEK.

The workflow consists of:

1. Hardware-dependent preprocessing.
2. Quantum time evolution and measurement.
3. Classical post-processing.
4. Kernel matrix construction.
5. SVM training and validation.

Infrastructure Implications This case study highlights several infrastructure-level considerations:

- The number of quantum shots scales with $O(N)$ per graph and $O(N^2)$ for full kernel matrix estimation.
- Parameter optimization multiplies hardware calls by the number of Bayesian iterations.
- Measurement failures and register initialization errors reduce effective shot counts.
- Large-scale graph embedding introduces a non-trivial classical preprocessing stage.

These aspects directly connect QML workloads to the execution models and resource management strategies discussed in Chapter 5.

The experimental evaluation of this case study is reported in Section 6.3. Classification performance is assessed on two nested graph subsets, `PROTEINS12` and `PROTEINS256`, across emulated and real-hardware execution on Aquila. The Bayesian-optimized waveform Λ_{BO} is compared against the reference schedule Λ_a and against the Shortest Path Kernel classical baseline. Key experimental questions include whether pulse optimization yields a measurable improvement in downstream F1-score, whether hardware execution degrades the resulting kernel relative to emulation, how robust the classification is under reduced shot budgets, and whether pulse parameters selected on the smaller `PROTEINS12` subset transfer effectively to the larger `PROTEINS256` evaluation.

3.5 Comparative Analysis of QML Paradigms

The previous sections introduced three families of Quantum Machine Learning approaches that are particularly relevant in the NISQ regime: variational quantum models and Quantum Neural Networks, Quantum Reservoir Computing, and quantum kernel methods (with a focus on distribution-based kernels such as the Quantum Evolution Kernel). While all these paradigms rely on a hybrid quantum–classical

execution model, they differ substantially in where trainability is "spent", which parts of the pipeline are quantum-critical, and how hardware constraints translate into algorithmic bottlenecks. Since the goal of this Chapter is to set the stage for the infrastructure-oriented discussions of Chapter 5 and the empirical results of Chapter 6, this section emphasizes the structural trade-offs rather than performance numbers.

A useful lens to compare NISQ QML paradigms is to separate (i) the *representation mechanism* provided by the quantum device from (ii) the *optimization mechanism* required to make that representation useful. Variational models employ a parameterized circuit family $U(\theta)$ and rely on a classical optimizer to adapt θ to a task-specific objective. The quantum processor therefore acts as a nonlinear feature generator whose configuration changes frequently across training iterations, and whose output is typically summarized by a small set of expectation values. In contrast, QRC shifts the burden of optimization away from the quantum device: the reservoir dynamics are fixed (or only weakly tuned), and learning is performed in a classical readout layer. As a consequence, QRC reduces the exposure to barren plateaus and decreases the amount of parameter search that must be performed under shot noise and hardware drift. Finally, quantum kernel methods eliminate end-to-end quantum training altogether: the quantum device is used to evaluate a similarity measure $K(\cdot,\cdot)$ induced by a feature map (explicit or implicit), after which the learning problem is solved classically (e.g., via SVMs). In this case, the "optimization" is largely displaced to classical hyperparameter selection (e.g., kernel bandwidth, regularization), and the quantum subroutine is repeated mainly because of dataset-size scaling rather than because of iterative parameter updates.

These distinctions have immediate implications for what limits scalability in practice. In variational QML, the dominant cost is often the number of circuit evaluations, driven by gradient estimation and optimizer choice, and compounded by the fact that each circuit must be compiled, executed with sufficient shots, and re-executed after parameter updates. This structure makes variational models sensitive to both quantum noise (which biases the objective) and classical–quantum orchestration overhead (which inflates wall-clock time). QRC, by contrast, is limited primarily by the reservoir size (qubit count and measurement bandwidth) and by the stability of the dynamical regime that produces useful separability in the readout space. Its key advantage is that it supports training workflows where the quantum device is operated in a more "streaming" fashion, often with repeated measurements and lightweight

classical updates. Quantum kernels occupy yet another corner: their quantum circuits can be kept shallow and hardware-native, but the number of evaluations scales at least quadratically with the number of data points in the naive setting, and additional outer-loop optimization (e.g., Bayesian tuning of evolution parameters in the QEK case study) multiplies this cost further.

From the standpoint of *hardware affinity*, the three paradigms also exhibit different "natural" mappings to platforms. Variational circuits are conceptually universal, but their effective depth is constrained by device connectivity and compilation overhead; hence, they are frequently optimized around hardware-efficient ansätze, which may sacrifice interpretability and can exacerbate trainability issues. QRC is particularly aligned with platforms that offer long-lived analog dynamics and fast measurement primitives, because the reservoir can be operated as a physical system rather than as a deep digital circuit. Kernel methods are often attractive on analog platforms as well, especially when the feature map is implemented via time evolution under a native Hamiltonian (as in QEK), enabling expressive transformations with limited circuit synthesis. In all cases, however, it is crucial to recognize that "hardware affinity" is not only a matter of physics: it also reflects how easily a workload can be batched, parallelized, and scheduled in the presence of queueing and latency.

To summarize these aspects concisely, Table 3.1 compares the three paradigms along algorithmic and infrastructural dimensions that will reappear in Chapter 5. On the other hand, Figure 3.3 reports schematics that abstracts each paradigm into a repeated pattern of quantum and classical stages, highlighting where iteration happens.

Across the three paradigms, the most consequential differences in the NISQ regime are not only about expressivity, but about *how algorithmic iteration translates into resource pressure*. Variational models create many short, latency-sensitive quantum invocations tightly coupled to an optimizer; QRC creates fewer configuration changes but sustained measurement workloads; kernels create a large number of shallow evaluations dominated by dataset-size scaling and by the engineering of efficient overlap/distribution estimation. These differences motivate the infrastructure-level discussion of the next section, which reframes QML not as a single workload type, but as a family of hybrid patterns that stress different parts of the QC–HPC stack.

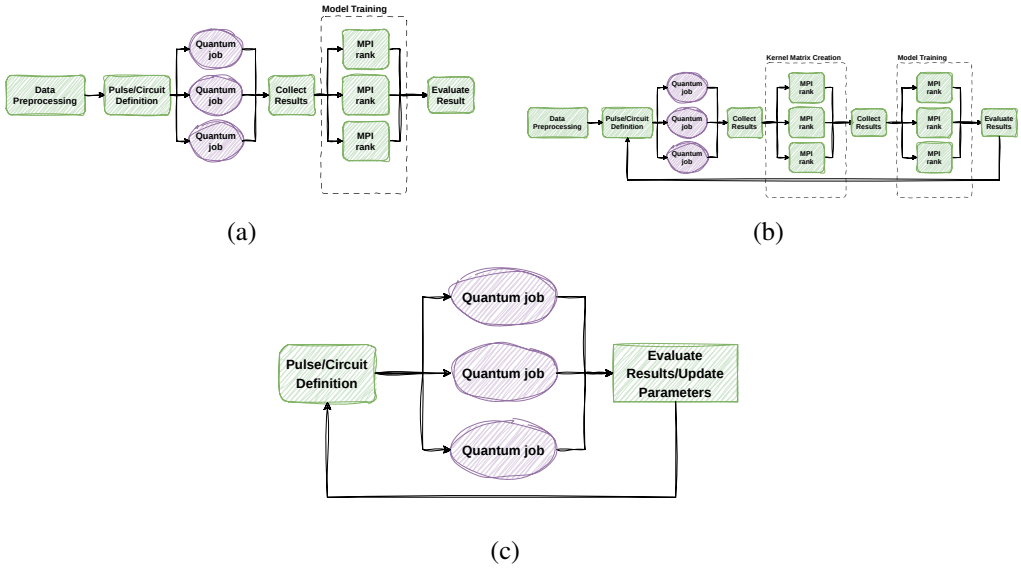


Fig. 3.3 Schematic comparison of execution patterns for QRC (a), quantum kernels (b), and variational QML (c).

3.6 Infrastructure-Level Implications of QML Workloads

Hybrid QML workloads are unusual from the perspective of traditional HPC scheduling because they combine (i) fine-grained and repeated interactions with a scarce accelerator (the QPU) with (ii) a classical control plane whose computational intensity may be modest compared to the orchestration overhead it induces. The three paradigms analyzed above stress the system in different ways, yet they share a set of infrastructure-level implications that can be made explicit and later mapped to the execution models and resource-management mechanisms developed in Chapter 5.

A first implication concerns the *latency-amplification effect* of iterative algorithms. In variational QML, the logical training loop is composed of many small quantum evaluations whose raw device time can be short compared to the end-to-end turnaround time (submission, queueing, compilation, execution, readout, post-processing). Since the optimizer requires sequential feedback, the wall-clock time is often determined by the slowest segment of this loop rather than by aggregate FLOPs. In practice, this means that queueing delay and control-plane latency can dominate training time even when the quantum circuit depth is shallow. The impact becomes

stronger whenever error-mitigation strategies increase shot counts and therefore enlarge the number of repeated evaluations required for each objective estimate.

A second implication is that QML exposes a tension between *throughput* and *synchronization*. Kernel methods are throughput-hungry because they demand large numbers of independent evaluations (e.g., pairwise kernel entries). This structure is highly parallelizable and can benefit from batched execution on the QPU side and distributed assembly on the classical side. At the same time, downstream classical steps (e.g., kernel matrix completion and SVM training) require global synchronization points. The overall performance therefore depends not only on peak throughput, but also on how efficiently the system can (i) keep the QPU busy with queued batched requests and (ii) deliver results to the classical aggregation stages without creating long idle periods. In QRC, the analogous tension is expressed as a dataflow problem: the QPU is repeatedly measured and produces a stream of classical features that must be ingested at rate, stored, and processed. Here, I/O and serialization costs can become central, especially when repeated measurements are used to expand the effective reservoir dimension.

A third implication relates to *resource asymmetry and blocking*. In an HPC-integrated setting, classical resources (nodes, cores, GPUs) are abundant relative to QPUs. If classical resources are allocated statically for the whole duration of a hybrid job, then any waiting time for QPU access translates into wasted node-hours. This phenomenon is particularly acute for workloads with intermittent quantum phases (which is typical of variational QML and some kernel pipelines). From a system-design perspective, this motivates mechanisms that decouple classical allocation from quantum access, either by splitting applications into workflow steps, by enabling dynamic resizing (malleability), or by virtualizing quantum access so that quantum idle time can be reduced without over-reserving classical nodes. While these mechanisms are developed later in the thesis, the important point here is that QML workloads provide a concrete, application-driven justification for them: the naive "reserve everything up front" model is often the least efficient for QPU-scarce settings.

A fourth implication is that *compilation and configuration overhead* must be treated as first-class costs. Variational QML changes circuit parameters frequently and can induce repeated compilation/transpilation work if not carefully engineered (e.g., by caching parameterized circuits, separating topology mapping from parameter

binding, and using runtime parameter updates when supported). For analog platforms (as in QRC and QEK), the analogous issue is pulse-programming and register preparation overhead, which can dominate if the quantum task is short but setup is expensive. In both cases, infrastructure-level solutions that support caching, warm starts, and persistent sessions are not merely conveniences; they can shift the effective regime in which a QML method is feasible.

A fifth implication is that QML is a natural producer of *multi-level parallelism*. At the algorithmic level, one can often parallelize across data batches, parameter candidates, cross-validation folds, and hyperparameter trials. At the infrastructure level, exploiting this parallelism requires a scheduler/runtime that can simultaneously (i) distribute classical work across nodes and (ii) multiplex quantum requests efficiently, while preserving the dependency constraints imposed by the algorithm (for instance, optimizer steps that require a consistent objective estimate). In other words, the scheduling problem is not simply "how to run many quantum jobs", but "how to coordinate a hybrid DAG with a scarce shared resource", which anticipates the workflow and reservation-based planning concepts discussed later. Table 3.2 maps each QML paradigm to the dominant infrastructure stressors and to the most relevant optimization knobs.

The core message is that QML workloads make the QC–HPC integration problem concrete: they simultaneously demand low-latency iteration (variational models), high-throughput quantum sampling (kernels and QRC), and careful avoidance of classical resource blocking due to QPU scarcity. Chapter 5 will formalize these observations by (i) defining execution patterns and synchronization models for hybrid applications, and (ii) introducing platform-aware strategies, including virtualization, workflow decomposition, reservation mechanisms, and malleability, that specifically target the inefficiencies revealed by QML-style control loops.

Table 3.1 Qualitative comparison of NISQ-relevant QML paradigms from an algorithmic and infrastructure perspective.

Dimension	**Variational / QNNs**	**QRC**	**Quantum kernels (QEK)**
Quantum role	Trainable map $U(\theta)$ evaluated iteratively	Fixed dynamical reservoir; classical readout only	Similarity via overlaps or distribution distances
Optimization locus	Classical optimizer tightly coupled to quantum loop (SPSA, COBYLA)	Classical readout fitting; optional light dynamics tuning	Classical training on kernel matrix; optional outer loop
Trainability risks	Barren plateaus; noise-biased objectives; optimizer instability	Reduced barren-plateau exposure; depends on regime and observability	No quantum training; sensitivity through kernel estimation noise
Quantum depth	Shallow–moderate; grows with ansatz and embedding overhead	Shallow per step; relies on repeated evolution	Shallow; often hardware-native
Scaling bottleneck	Circuit evaluations per iteration $\times$ iterations	Reservoir size and measured observables; streaming rate	$O(N^2)$ kernel evaluations; shot budget
Noise sensitivity	High; mitigation increases sampling cost	Moderate; averaging over shots	Moderate; distribution-based kernels smooth fluctuations
Parallelism	Parameter- and shot-level; batching essential	Sample/batch-level; streaming-friendly	Pairwise evaluation; outer-loop tuning
Best-fit platforms	Broad; depends on connectivity and compilation	Analog-friendly; fast measurement	Native evolution (analog) or efficient overlap estimation
Orchestration pattern	Tight synchronous loop	Streamed acquisition + readout fitting	Two-phase: kernel acquisition, then classical training

Table 3.2 Infrastructure stressors and practical optimization knobs for NISQ QML paradigms.

Paradigm	**Dominant stressors**	**Practical knobs (system & runtime)**
Variational QML / QNNs	Latency and queueing in tight feedback loops; high circuit-evaluation counts; compilation overhead	Batching of parameter evaluations; circuit caching and late parameter binding; asynchronous objective estimation where possible; shot allocation policies; co-scheduling strategies to reduce classical idling
QRC	Sustained measurement throughput; data movement and storage; repeated measurements as feature expansion	Streaming-oriented execution; aggregation close to data source; controlled shot budgeting; persistent device sessions; efficient feature extraction pipelines
Quantum kernels (e.g., QEK)	Large number of independent evaluations (often $O(N^2)$); outer-loop hyperparameter search; global synchronization to assemble kernel	Batched kernel acquisition; distributed kernel assembly; aggressive parallelization of pairwise computations; reuse of partial kernels across hyperparameter trials; scheduling policies prioritizing throughput over latency

Chapter 4

Hybrid Quantum Approaches to Combinatorial Optimization

Building upon the conceptual and technological foundations discussed in Chapter 2, this chapter focuses on the design, implementation, and platform-aware optimization of hybrid quantum–classical workflows for combinatorial optimization problems. While Chapter 2 introduced the theoretical background of variational algorithms, quantum annealing paradigms, and hybrid integration models in a technology-agnostic manner, the present chapter narrows the scope to practical algorithmic frameworks and their deployment on NISQ devices.

4.1 Combinatorial Optimization in the NISQ Era

Combinatorial optimization represents one of the most promising near-term application domains for quantum computing. Many relevant industrial and scientific tasks can be formulated as discrete optimization problems over binary or integer variables, often characterized by NP-hard complexity. Examples include routing and scheduling, resource allocation, portfolio optimization, and graph-theoretic problems such as maximum cut and graph partitioning. In the NISQ regime, fully fault-tolerant algorithms are not yet available, and hybrid approaches that combine classical optimization loops with parameterized quantum circuits have emerged as a pragmatic strategy.

4.2 QUBO and Ising Formulations

Let us consider a generic combinatorial optimization problem expressed as

$$\min_{x\in\{0,1\}^n} f(x), \tag{4.1}$$

where $f : \{0,1\}^n \to \mathbb{R}$ is a cost function encoding the objective and possibly constraints. A broad class of such problems can be reformulated as a Quadratic Unconstrained Binary Optimization (QUBO) problem,

$$\min_{x\in\{0,1\}^n} x^\top Qx + c^\top x, \tag{4.2}$$

with $Q \in \mathbb{R}^{n\times n}$ symmetric and $c \in \mathbb{R}^n$. Constraints are typically incorporated through penalty terms, as discussed in Section 2.4.

The QUBO formulation is particularly suitable for quantum implementations because it can be mapped to an Ising Hamiltonian of the form

$$H_C = \sum_i h_i Z_i + \sum_{i<j} J_{ij} Z_i Z_j, \tag{4.3}$$

after the variable transformation $x_i = (1 - Z_i)/2$, where Z_i denotes the Pauli-Z operator acting on qubit i. This cost Hamiltonian H_C encodes the objective function in its ground state, establishing the link between combinatorial optimization and quantum energy minimization.

While this mapping is standard and has been extensively described in Chapter 2, here we emphasize the implementation implications. In practice, the structure of Q determines the required two-qubit interactions. Sparse instances lead to shallow circuits and reduced embedding overhead, whereas dense problems impose heavy connectivity requirements that must be reconciled with the hardware topology. This aspect becomes critical when targeting superconducting devices with limited connectivity or neutral-atom arrays with geometric constraints.

4.3 Variational Quantum Optimization

Among the hybrid paradigms introduced in Chapter 2, the QAOA plays a central role. It alternates between unitary evolutions generated by the cost Hamiltonian H_C and a mixing Hamiltonian $H_M = \sum_i X_i$, producing a parameterized state

$$|\psi(\gamma,\beta)\rangle = \prod_{l=1}^{p} e^{-i\beta_l H_M} e^{-i\gamma_l H_C} |+\rangle^{\otimes n}, \tag{4.4}$$

where p is the circuit depth and (γ,β) are variational parameters optimized via a classical routine.

In contrast to the general overview provided in Section 2.4, the focus here is on practical considerations:

- parameter initialization strategies (random, heuristic warm-start, or classical pre-optimization);
- choice of classical optimizers (e.g., COBYLA, SPSA, gradient-based methods with parameter-shift rules);
- shot allocation and statistical error mitigation;
- hardware-native gate decompositions and depth reduction techniques.

The classical outer loop typically minimizes the expectation value

$$E(\gamma,\beta) = \langle\psi(\gamma,\beta)|H_C|\psi(\gamma,\beta)\rangle, \tag{4.5}$$

estimated through repeated measurements. As discussed in Chapter 2, noise and barren plateaus can significantly affect convergence, especially for increasing p.

Alternative variational approaches, such as hardware-efficient ansätze and adaptive variants (e.g., ADAPT-QAOA), have also been considered in the literature. However, in this thesis we primarily focus on structured ansätze that reflect the problem Hamiltonian, as they offer improved interpretability and reduced parameter count, which is particularly relevant in the NISQ regime.

4.3.1 Hardware Constraints and Embedding

A critical step in translating a QUBO or Ising model into a quantum circuit is the embedding onto the physical qubit connectivity graph. Depending on the platform, this may involve:

- SWAP insertion for limited connectivity (superconducting qubits);
- geometric placement constraints (neutral-atom arrays);
- minor embedding with chain variables (annealing architectures).

Connectivity-aware compilation strategies and routing algorithms significantly influence the final circuit depth. As shown in recent compiler-level studies (e.g., [34, 104]), hardware-aware transpilation can improve performance by reducing two-qubit gate counts and aligning operations with high-fidelity links.

In the context of this thesis, problem instance structure, ansatz design, and compiler strategies are jointly optimized to minimize hardware overhead.

4.4 Hybrid Decomposition and Classical-Quantum Cooperation

A practical hybrid optimization workflow consists of several layers, spanning from high-level problem definition to low-level hardware execution. The typical pipeline includes:

1. classical preprocessing and QUBO construction;
2. circuit generation and parameterization;
3. compilation and hardware mapping;
4. quantum execution and measurement;
5. classical post-processing and parameter update.

Although conceptually simple, this pipeline becomes computationally demanding when executed at scale. Each variational iteration may require thousands of circuit evaluations, and large hyperparameter sweeps are often necessary to assess performance across different depths, optimizers, and noise conditions.

For this reason, the integration with HPC resources becomes essential. In this thesis, hybrid orchestration strategies are employed to distribute circuit evaluations, manage job queues, and collect measurement statistics efficiently. This integration is aligned with the HPC stack discussed in Section 2.5, where quantum processors are treated as accelerators within a larger heterogeneous computing infrastructure.

4.4.1 Parallelization and Resource Management

From a computational standpoint, several levels of parallelism can be exploited:

- parameter-level parallelism (evaluating different parameter sets concurrently);
- shot-level parallelism (distributing repeated measurements);
- instance-level parallelism (running multiple problem instances).

Efficient resource management requires careful scheduling, especially when accessing cloud-based quantum backends with queue constraints. Middleware layers can abstract hardware heterogeneity, enabling uniform submission of experiments across simulators and physical devices. This aspect is particularly relevant when comparing results across platforms, as discussed in later Sections of this chapter.

Recent studies on hybrid orchestration frameworks (e.g., [105, 106]) highlight the importance of minimizing latency between classical and quantum components. In practice, batching strategies and asynchronous job handling significantly reduce turnaround time and improve optimizer stability.

4.4.2 Error Mitigation and Performance Metrics

Given the NISQ setting, error mitigation plays a crucial role. Techniques such as zero-noise extrapolation, measurement error mitigation, and probabilistic error

cancellation have been explored in the literature [30, 107]. In this thesis, lightweight mitigation techniques are favored to balance accuracy and computational overhead.

Performance is evaluated using metrics that go beyond raw energy expectation values. These include:

- approximation ratio with respect to the classical optimum;
- success probability of sampling optimal or near-optimal solutions;
- robustness to noise and parameter perturbations;
- scalability with problem size.

Such metrics provide a more comprehensive assessment of practical utility, particularly when comparing quantum-enhanced methods with state-of-the-art classical solvers.

4.5 Case Study III: BBQ-mIS: Solving the Graph Coloring Problem

This Section presents a hybrid quantum–classical workflow for solving graph coloring problems through a decomposition into Maximum Independent Set (MIS) subproblems [108]. The methodology is based on the BBQ-mIS algorithm, originally developed by the author of this dissertation in collaboration with co-authors and presented in [6]. The work constitutes one of the core original contributions of this thesis in the domain of combinatorial optimization and HPC–QC integration. While Chapter 2 reviewed general quantum approaches to graph coloring and QUBO-based formulations, the present Section focuses on the design choices, architectural implications, and integration aspects that emerged from the development of BBQ-mIS. The experimental evaluation and quantitative assessment are deferred to Chapter 6.

4.5.1 Problem Formulation

Given an undirected graph $G = (V, E)$, the Graph Coloring (GC) [109] problem consists in assigning a color to each vertex such that adjacent vertices receive

different colors, while minimizing the total number of colors used. The minimum number of colors required is the chromatic number $\chi(G)$. The problem is NP-hard and arises in multiple application domains, including analysis of biological networks, scheduling, resource allocation, network planning and register allocation [110–112, 12].

A direct encoding of GC into a QUBO formulation typically requires $|V| \times k$ binary variables for a k-color representation, leading to a rapid growth in problem size that is incompatible with current NISQ hardware constraints. In this work, a different strategy was pursued, exploiting the structural relation between graph coloring and Independent Sets.

An Independent Set is a subset of vertices with no internal edges, and a MIS is an Independent Set of maximum cardinality. Since all vertices in an Independent Set can share the same color, GC can be reformulated as an iterative removal of Independent Sets from the graph. This observation motivated the development of a hybrid decomposition strategy, where MIS instances are delegated to a quantum subroutine, while the global coloring logic remains classical.

4.5.2 Algorithmic Design of BBQ-mIS

The BBQ-mIS algorithm was designed by the author to combine three tightly integrated components:

- a quantum-enhanced MIS solver,
- a classical Branch-and-Bound (B&B) framework for color minimization,
- a parallel execution model tailored to HPC environments.

The overall structure is illustrated in Figure 4.1. Instead of solving GC in a monolithic manner, the problem is decomposed into a sequence of MIS subproblems, each compatible with a binary encoding and thus suitable for quantum treatment.

A preliminary greedy iterative strategy, denoted Greedy-it-MIS in the original paper, was first implemented as a baseline. However, although this approach guarantees feasibility, it does not ensure minimal color usage. The central original contribution

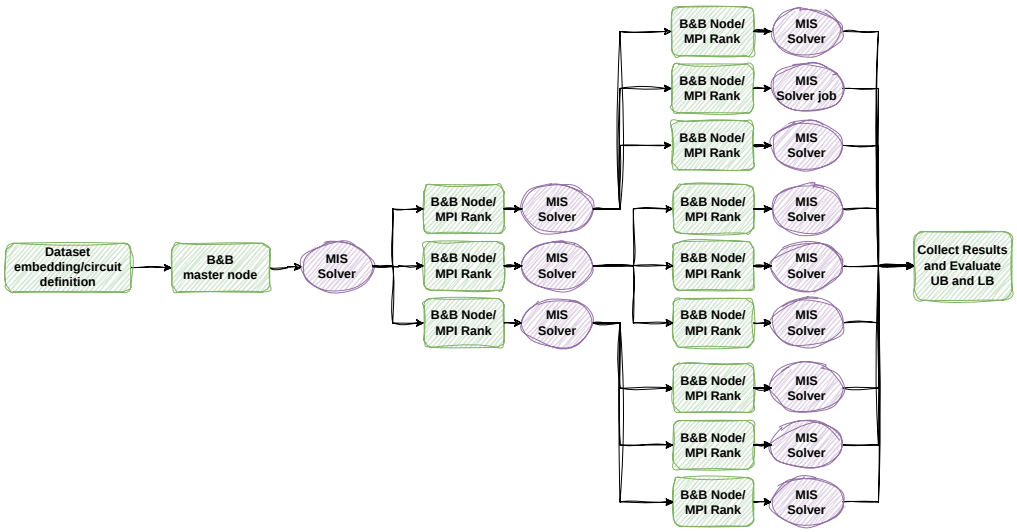


Fig. 4.1 Visualization of the hybrid workflow of the BBQ-mIS algorithm. Green boxes represent the classical parts of the workflow, while purple ovals the quantum ones. The graph coloring problem is decomposed into MIS subproblems solved via a quantum solver, while a classical Branch-and-Bound layer orchestrates the global search.

was therefore the introduction of a Branch-and-Bound layer that systematically explores alternative sequences of maximal Independent Sets (mISs). A mIS is defined as an Independent Set, which is not properly contained in another Independent Set of the graph G, i.e., adding one or more vertexes to a mIS would result in losing its independence. Indeed, the theorem whose proof is reported in [6] guarantees that every graph $G = (V,E)$ has an optimal coloring in which (at least) one of the colors is a mIS. Hence, we can restrict our solutions space investigation to mISs.

4.5.3 Quantum MIS Subroutine

Each MIS instance is expressed as a binary combinatorial optimization problem. Given an induced subgraph $H = (U,I)$ at a node of the B&B tree, the MIS can be encoded through a QUBO formulation of the form

$$\min_{x\in\{0,1\}^{|U|}} \left(-\sum_{i\in U} x_i + \lambda \sum_{(i,j)\in I} x_i x_j \right), \tag{4.6}$$

where $x_i = 1$ selects vertex i and λ penalizes violations of independence constraints.

In the original implementation, the mapping to quantum hardware exploited the natural suitability of neutral-atom architectures for MIS problems, where Rydberg blockade effects intrinsically encode exclusion constraints. This platform-aware choice was motivated by the desire to minimize encoding overhead and qubit usage. The hardware Hamiltonian effectively implements a physical realization of an MIS-like objective, thereby reducing compilation complexity. This allow to implement both QAOA and Quantum Annealing Algorithm (QAA) [113].

At the same time, the algorithm was designed to remain conceptually portable to gate-based platforms, where the QUBO can be embedded within a QAOA-type variational circuit, as discussed in Section 4.3. Indeed, a superconducting-based implementation has been used for evaluating the classical–quantum integration solution proposed in Section 5.3. The details and relative results are reported in Section 5.4.

However, it is important to note that, as for the Case Study II described in Section 3.4.4, solving the MIS problem using neutral atom simulators requires graphs to satisfy some geometrical requirements. Therefore, also in this case, we implemented a graph embedding step (see Appendix A.1).

An important methodological insight emerging from the author's work concerns the use of measurement histograms. Instead of selecting only the most frequent solution returned by the quantum routine, multiple mISs are extracted and passed to the classical layer. This choice enriches the branching space of the B&B tree and directly leverages the intrinsic sampling nature of quantum hardware.

4.5.4 Branch-and-Bound Formulation

The core innovation of BBQ-mIS lies in embedding the MIS solver within a Branch-and-Bound scheme. Each node of the B&B tree is associated with a state

$$(H, C, k, LB, UB),$$

where H is the remaining subgraph, C the partial coloring, k the number of colors used so far, and LB and UB are lower and upper bounds on the achievable chromatic number from that node.

The bounding mechanism combines spectral lower bounds (e.g., Hoffman-type bounds) and degree-based upper bounds to prune non-promising branches. In addition to classical pruning by non-improvement, the algorithm incorporates problem-specific pruning rules that discard non-maximal or non-Independent Sets returned by the quantum solver. Furthermore, a fingerprinting strategy avoids redundant exploration of identical induced subgraphs generated through different branching orders.

A schematic representation of the B&B structure is shown in Figure 4.2. The integration of quantum-generated mISs into a classical optimization scaffold reflects a deliberate design choice: optimality guarantees and global search control remain classical, while the quantum device acts as a combinatorial candidate generator.

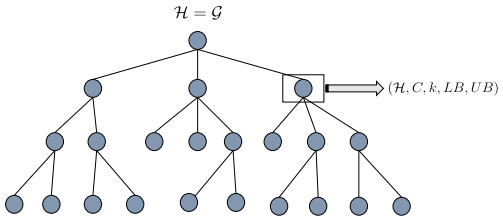
4.5.5 Parallelization and HPC Considerations

A further original contribution of this work concerns the parallelization of the BBQ-mIS algorithm and its deployment in an HPC environment. The Branch-and-Bound structure naturally supports parallel exploration of different nodes. An MPI-enhanced implementation was developed to distribute subtrees across multiple workers, each invoking the MIS quantum routine independently.

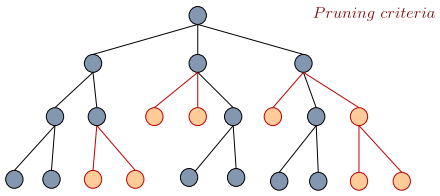
This design explicitly addresses the asymmetry between abundant classical resources and scarce quantum devices. Multiple classical processes may generate concurrent requests to the same QPU, making queueing effects and synchronization critical performance factors. These characteristics make BBQ-mIS an ideal testbed for the execution models and virtualization mechanisms discussed in Chapter 5.

4.5.6 Methodological Relevance within the Thesis

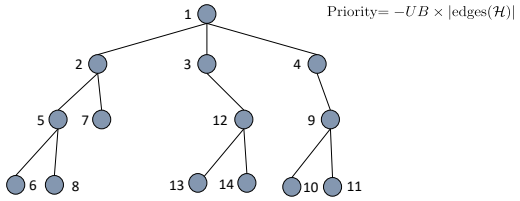
From a cross-layer perspective, BBQ-mIS represents a concrete realization of the co-design philosophy advocated in this dissertation. The algorithm was not conceived in isolation at the quantum layer; rather, its structure reflects:



(a) Each node in the Branch&Bound tree is associated with the induced graph $\mathscr{H}$, the corresponding coloring C, the number of colors k, a lower bound LB on the best coloring achievable by branching further on that node, and an upper bound UB on $\chi(\mathscr{H})$.



(b) The pruning criteria reduce the number of Branch&Bound nodes to explore taking into account non-improving directions, unfeasibility, and equivalences of graphs $\mathscr{H}$.



(c) A priority score is assigned to each Branch&Bound node to lead the exploration of the Branch&Bound tree: higher priority scores mean that the corresponding node is explored sooner.

Fig. 4.2 Representation of the Branch-and-Bound scheme underlying the BBQ-mIS algorithm.

- hardware-aware encoding choices,
- explicit separation of quantum sampling and classical optimization,
- compatibility with distributed HPC execution,
- sensitivity to latency and resource contention.

The experimental evaluation of BBQ-mIS is reported in Section 6.4 and proceeds in two phases. First, the Branch-and-Bound decomposition is validated in emulation on a benchmark of 120 unit-disk graph instances with 10–15 vertices, comparing BBQ-mIS colorings against optimal solutions obtained from an exact classical solver. Second, three representative MIS subproblems—extracted from successive nodes of the Branch-and-Bound search path of a 20-node instance—are executed on Aquila

with approximately 900 valid shots per task; the resulting mIS distributions are compared against classical emulation using 128-shot batches. The central experimental questions are whether the Branch-and-Bound decomposition achieves optimal graph colorings under emulated quantum sampling, and whether the real neutral-atom device preserves enough distributional structure to supply combinatorially useful mIS candidates. The present Section highlights instead the architectural and algorithmic decisions that motivated the design choices, substantiating the broader thesis argument: in the NISQ era, hybrid combinatorial optimization must be approached through coordinated cross-layer considerations, from problem encoding to infrastructure integration.

4.6 Comparative Analysis of Hybrid Optimization Paradigms

The previous Sections introduced the main algorithmic families through which near-term quantum devices are currently explored for combinatorial optimization, with particular emphasis on hybrid strategies that explicitly acknowledge NISQ constraints. In this context, "hybrid" does not only mean that a classical optimizer is present, but rather that a classical algorithmic scaffold is responsible for problem decomposition, feasibility handling, and (when required) global optimality guarantees, while the quantum device contributes either (i) approximate candidate solutions, (ii) samples from structured distributions, or (iii) estimates of objective-related quantities. This Section consolidates these ideas into a comparative view of hybrid optimization paradigms, focusing on where the quantum component is placed in the optimization loop, which resources dominate cost, and how platform characteristics influence practical viability.

A first paradigm is represented by *variational gate-based optimization*, of which the Quantum Approximate Optimization Algorithm is the canonical example. QAOA alternates between a problem Hamiltonian H_C encoding the cost function and a mixing Hamiltonian H_M, producing a parameterized state

$$|\psi(\gamma,\beta)\rangle = \left(\prod_{\ell=1}^{p} e^{-i\beta_\ell H_M} e^{-i\gamma_\ell H_C}\right) |+\rangle^{\otimes n}, \tag{4.7}$$

whose measurement statistics induce a distribution over bitstrings; the classical outer loop tunes (γ, β) to minimize (or maximize) the expected cost [64, 114]. From a methodological standpoint, QAOA is appealing because it provides a systematic bridge between continuous control parameters and discrete solutions, and because the depth p offers an explicit expressivity knob. In practice, however, QAOA inherits many of the execution characteristics discussed for variational QML: it requires repeated circuit evaluations under shot noise, it can be sensitive to optimizer stability, and it often depends critically on compilation quality and connectivity-aware circuit synthesis. As a consequence, QAOA-based pipelines are typically designed around shallow depths, parameter-transfer heuristics, warm-start strategies, and problem-specific encodings that reduce circuit resources at the expense of generality.

A second paradigm is *analog Ising/QUBO optimization*, most visibly instantiated through quantum annealing and related adiabatic-inspired protocols. Here, the combinatorial problem is mapped to an Ising Hamiltonian or QUBO form, typically using binary variables $x_i \in \{0,1\}$ (or spins $s_i \in \{\pm 1\}$) such that the objective and constraints are represented by quadratic terms, possibly after introducing slack variables and penalty weights [115]. The quantum device implements a physical evolution (anneal) intended to concentrate probability mass on low-energy configurations. From an algorithmic point of view, this paradigm trades the iterative parameter tuning of QAOA for repeated sampling of a hardware-defined process, where performance depends strongly on embedding overhead, calibration, and the relationship between the problem energy landscape and the device dynamics [116]. In hybrid settings, annealing is rarely used in isolation; rather, it is integrated into classical metaheuristics, decomposition frameworks, or exact methods as a subroutine that proposes good candidates (primal heuristics) or improves incumbents, while the classical layer maintains feasibility and orchestrates restarts and parameter sweeps.

A third paradigm comprises *quantum-assisted classical exact or decomposition methods*. The BBQ-mIS case study described earlier in this chapter is representative: the global search and optimality guarantees are handled classically via Branch-and-Bound, while a quantum subroutine is invoked to generate high-quality mISs, which are then injected into a column-generation-like coloring strategy and used to accelerate bounding and pruning. More generally, quantum subroutines can be embedded into Branch-and-Bound, Branch-and-Cut, Branch-and-Price, or Benders decomposition frameworks as (i) pricing or separation oracles, (ii) primal heuristics, or (iii) approximate solvers for NP-hard subproblems that dominate runtime [117,

118]. The conceptual advantage of this paradigm is structural: it makes the quantum component optional and opportunistic, and it preserves a clear correctness story because the classical framework can fall back to purely classical routines when the quantum solver does not deliver useful information. The practical challenge is that the resulting workflow is often *heterogeneous and bursty*: quantum calls occur at irregular times, depend on the evolving state of the search tree, and may involve many small subinstances rather than a single monolithic QUBO.

These paradigms can be compared through a resource-oriented lens. In QAOA and related variational gate-based methods, the dominant cost is typically the number of circuit evaluations required by the classical tuning loop, multiplied by the shot budget needed to obtain sufficiently low-variance objective estimates. In analog annealing, the cost is often driven by repeated runs under different embeddings, gauge transformations, anneal schedules, or penalty weights, combined with classical post-processing and feasibility repair. In quantum-assisted exact/decomposition methods, the cost depends less on a single tunable loop and more on how frequently and how effectively the quantum routine can contribute to the classical method (e.g., by tightening bounds, improving incumbents, or producing diverse columns/cuts). This is also the paradigm in which infrastructure effects become most visible: even a strong quantum subroutine can be neutralized if latency, queueing, or contention prevent it from being invoked at the right points of the classical search.

Table 4.1 summarizes the comparison in a way that is directly aligned with the thesis narrative, emphasizing the placement of the quantum role, the scaling bottlenecks, and the implications for cross-layer co-design. Similarly to what has been done for QML, Figure 4.3 shows abstracted hybrid workloads for each considered optimization paradigm.

The overarching message is that “hybrid optimization” is not a single workload class. Rather, it is a family of patterns that differ in whether quantum compute is a latency-critical component (as in iterative variational tuning), a throughput-critical component (as in large-scale sampling for annealing), or an *opportunistic accelerator* embedded within a correctness-preserving classical algorithm (as in BBQ-mIS-style designs). This distinction is essential for the remainder of the thesis, because it determines which execution models and resource-management strategies can realistically convert quantum device time into end-to-end application benefit.

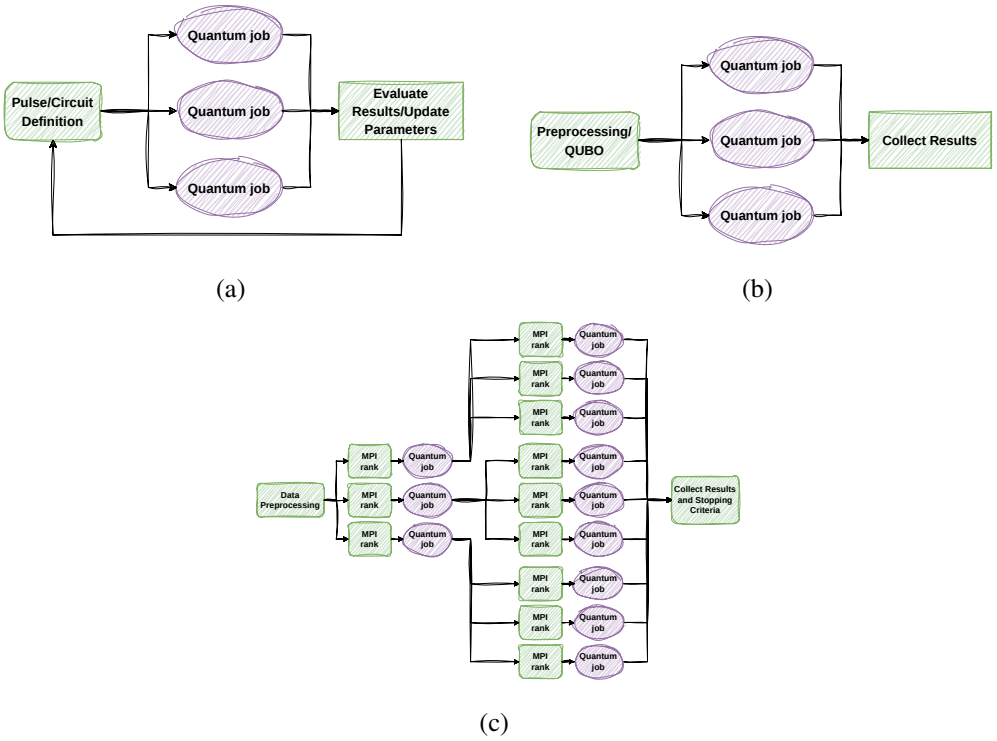


Fig. 4.3 Schematic comparison of execution patterns for hybrid optimization: (a) variational tuning loops (QAOA-like), (b) throughput-oriented annealing sampling and (c) quantum-assisted exact/decomposition methods with irregular, state-dependent quantum invocations.

4.7 Infrastructure-Level Implications of Optimization Workloads

Hybrid combinatorial optimization workloads stress the QC–HPC stack in ways that differ from QML, despite superficial similarities such as repeated quantum invocations and classical outer loops. The defining feature of optimization workloads is that the quantum component is frequently used as a *discrete sampler* over feasible (or near-feasible) candidates, and therefore the system must support both high-throughput sampling and timely delivery of candidate solutions to the classical decision logic. In addition, the control structure of exact and decomposition methods introduces irregularity: the sequence of subproblems is data-dependent, and the "value" of a quantum call depends on its arrival time in the search process. This Section translates these observations into infrastructure-level implications that will be formalized and revisited through the platform-aware strategies in Chapter 5.

A first implication concerns *latency sensitivity under adaptive control*. Variational optimization (QAOA-like) exhibits a tight feedback loop similar to variational QML: objective evaluations and parameter updates are sequentially coupled, and delays inflate wall-clock time multiplicatively. However, adaptive classical exact/decomposition methods introduce a subtler latency dependence: quantum calls are triggered at specific decision points (e.g., when generating promising Independent Sets for a node, or when solving a QUBO master problem in a decomposition iteration). If the quantum result arrives too late, the classical algorithm may have already explored alternative branches, updated incumbents, or tightened bounds using purely classical heuristics, thereby reducing the marginal benefit of the delayed quantum contribution. In other words, for these workloads, latency affects not only *time-to-solution* but also *algorithmic relevance*.

A second implication is the need to manage *contention and multiplexing* explicitly. Optimization workloads naturally generate many quantum requests: in sampling-heavy paradigms, this is due to the number of restarts and parameter sweeps; in tree-based paradigms, it is due to the number of subinstances encountered across nodes and across parallel workers. When multiple classical processes issue concurrent requests to a scarce QPU, naive queueing can lead to pathological behavior: workers block, classical resources are wasted, and the QPU becomes a bottleneck that dominates global scaling. The BBQ-mIS parallelization discussed earlier illustrates this effect in a concrete way, and it motivates the thesis emphasis on orchestration mechanisms that can limit blocking, support batching, and provide fairness and prioritization policies for quantum requests.

A third implication is the importance of *batching without breaking adaptivity*. For annealing-style sampling and kernel-like pairwise computations, batching is straightforward because evaluations are largely independent. For Branch-and-Bound or decomposition frameworks, batching is more delicate because subproblems are generated dynamically and because the classical logic often expects prompt feedback. Nevertheless, infrastructure-aware batching can still be beneficial if designed around short horizons: for instance, a worker can accumulate a small queue of subinstances (or multiple embeddings of the same subinstance) and submit them as a single batched quantum job, trading a modest increase in per-result latency for a large reduction in per-call overhead. This logic parallels the caching and late-binding strategies used for variational circuits, but must be adapted to the combinatorial setting where problem instances, not just parameters, change across calls.

A fourth implication relates to *state persistence and compilation/embedding amortization*. In gate-based optimization, repeated transpilation of structurally similar circuits can be reduced by caching parameterized circuits and using runtime binding. In annealing and QUBO-based workflows, the analog of compilation is the embedding/mapping step, as well as the selection of penalty weights and scaling factors that balance objective and constraints. Since these steps may be costly and instance-dependent, the system benefits from persistent sessions that keep device context warm, reuse embeddings across related subinstances, and avoid reinitializing the full software stack for every small QUBO. The net effect is that the boundary between "algorithm" and "system" becomes blurred: embedding reuse and session persistence are, in practice, algorithmic accelerators.

A fifth implication is that optimization workloads frequently produce *hybrid DAGs with mixed granularity*. A single run may include phases of intensive classical computation (bounding, cut separation, repair heuristics) interleaved with bursts of quantum sampling. This structure suggests that static resource allocation is inefficient: reserving a full classical allocation throughout the entire run can lead to substantial idle time when the workflow is stalled on quantum access, while reserving QPU access continuously is impossible due to scarcity and multi-tenancy. This observation connects directly to the need for execution models that support workflow decomposition, elastic resource usage, and explicit synchronization semantics, as discussed later in the thesis and in the hybrid workflow literature [119].

Table 4.2 condenses these implications into a system-oriented mapping from paradigm to stressor and to practical runtime knobs. The table is intended to complement Table 4.1 by shifting perspective from algorithm design to execution and resource management.

In conclusion, the infrastructure implications of hybrid combinatorial optimization can be summarized as follows. First, performance depends as much on orchestration and resource contention as on the quality of the quantum sampler, because optimization pipelines are frequently adaptive and time-sensitive. Second, the "right" execution model differs by paradigm: variational methods require latency-aware tight loops; annealing-style methods require throughput-oriented sampling services; quantum-assisted exact/decomposition methods require workflow-level integration with prioritization, fallbacks, and careful management of blocking. These obser-

vations provide a direct motivation for the execution patterns and for the platform-aware integration strategies developed in Chapter 5.

Table 4.1 Qualitative comparison of hybrid paradigms for combinatorial optimization in the NISQ era.

Dimension	**Variational gate-based (e.g., QAOA)**	**Analog Ising/QUBO (e.g., annealing)**	**Quantum-assisted exact/decomposition**
Quantum role	Parameterized sampler; probability mass shaped by (γ, β)	Physical sampler of low-energy states under an anneal protocol	Candidate generator / oracle within a classical global method (B&B, Benders, CG, ...)
Optimization locus	Tight hybrid loop: classical optimizer tunes circuit parameters	Mostly classical orchestration: sweeps over embeddings/schedules/penalties; post-processing	Classical framework controls correctness; quantum invoked opportunistically for subproblems
Primary bottleneck	Circuit-evaluation count $\times$ shots $\times$ iterations; compilation overhead	Embedding overhead; repeated sampling and parameter sweeps; constraint penalties	Latency/queueing and irregular invocation patterns; quality and diversity of quantum outputs
Noise sensitivity	High: impacts objective estimates and parameter search	Moderate: manifests as sampling bias and limited effective precision	Moderate: classical method can absorb failures, but timing and consistency matter
Parallelism opportunities	Batch evaluations (shots, parameter sets) and multi-starts	Massive sampling parallelism; gauge/seed parallelism	Subtree and subinstance parallelism; distributed workers issuing quantum requests
Best-fit integration pattern	Latency-aware sessions; caching and batched execution; parameter transfer	Throughput-oriented sampling service with efficient embedding pipelines	Workflow/DAG integration with adaptive calls, fallbacks, and contention management

Table 4.2 Infrastructure stressors and practical optimization knobs for hybrid combinatorial optimization workloads.

Paradigm	**Dominant stressors**	**Practical knobs (system & runtime)**
Variational (QAOA-like)	Latency amplification in iterative loops; compilation overhead; shot noise	Circuit caching and late parameter binding; batched evaluations/multi-starts; session-based execution; adaptive shot allocation
Analog Ising/QUBO (annealing-like)	Throughput demand from sampling and sweeps; embedding and penalty tuning; data movement	Batched sampling jobs; embedding reuse/caching; structured parameter sweeps; fast post-processing and feasibility repair pipelines
Quantum-assisted exact/decomposition	Irregular and state-dependent quantum calls; contention across workers; blocking of classical resources	Priority-aware request scheduling; bounded batching windows; asynchronous invocation with fallbacks; co-scheduling to reduce idle classical time; admission control for quantum requests

Chapter 5

Efficient Resource Management for Hybrid QC–HPC Systems

This Chapter analyzes the architecture of quantum platforms installed on-premise in research and computing centers, identifies requirements for their integration within production HPC environments, and proposes resource management strategies matched to the hybrid workflow taxonomy established in Chapters 3 and 4.

In particular, the work described in the following Sections is the product of the *SmartHPCQC* project[1] performed in collaboration between Politecnico di Torino, LINKS Foundation, Università di Torino, E4 Engineering and CINECA. As a final note, the research described here has already been published [7, 8, 3].

5.1 Hybrid Execution in the NISQ Era

The NISQ regime is commonly framed in terms of hardware-facing constraints (limited qubit counts, non-negligible noise, calibration drift, and restricted connectivity) which bound circuit depth and achievable fidelity. While these factors remain fundamental, the hybrid quantum–classical methods targeted in this thesis are often limited, in practice, by *execution efficiency* rather than by circuit expressivity alone. Hybrid workloads for QML and combinatorial optimization typically comprise repeated invocations of short quantum kernels embedded in classical control loops.

[1] funded by the European Union NextGenerationEU, ICSC National Centre, CN00000013, MUR Act n. 1031 - 17/06/2022

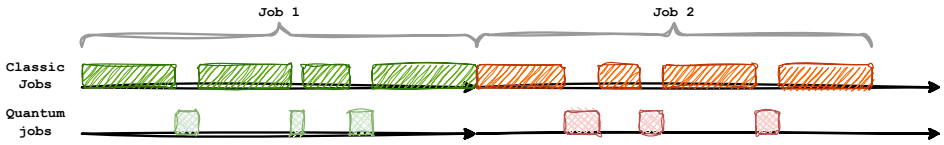


Fig. 5.1 Sequential execution of two hybrid jobs in a multi-tenant, single QPU scenario on a hybrid HPC–QC cluster. Job 2 cannot begin its classical phase until Job 1 releases the QPU, leaving both the QPU and classical nodes idle for extended intervals.

Consequently, end-to-end performance depends on a tightly coupled set of cross-layer effects, including queueing and admission control on shared QPUs, classical–quantum synchronization points, batching and shot management, compilation and submission overheads, and system-level scheduling policies governing multi-user access.

A recurrent operational friction point stems from the access model of current QPUs. Quantum devices are usually exposed via vendor-managed client–server interfaces and proprietary queuing systems, whereas HPC environments are governed by batch schedulers, allocation policies, and formal accounting. Even when both worlds can be connected at the API level, this mismatch directly impacts time-to-solution: naive execution strategies can leave classical resources idle during long quantum phases or leave the QPU underutilized when quantum calls are short and interleaved with substantial classical work. The impact becomes particularly severe in shared, multi-tenant settings, where exclusive access to a single QPU serializes hybrid jobs and throttles throughput even if classical resources remain available, as shown in Figure 5.1.

A key driver of these inefficiencies is the *imbalance of time scales* between classical and quantum phases and, importantly, the fact that quantum service times are strongly shaped by the hardware modality. For superconducting platforms, typical circuit executions can be comparatively fast, making the overheads of compilation, network transfer, and queueing a significant fraction of end-to-end latency. In contrast, platforms such as neutral atoms may exhibit longer end-to-end quantum phases due to state preparation and (re)configuration, so that co-scheduling decisions must account for prolonged periods in which either the classical side or the quantum side can be waiting. Figure 5.2 shows the time scales in hybrid QC–HPC execution for different quantum platforms in relation to the typical granularity of SLURM

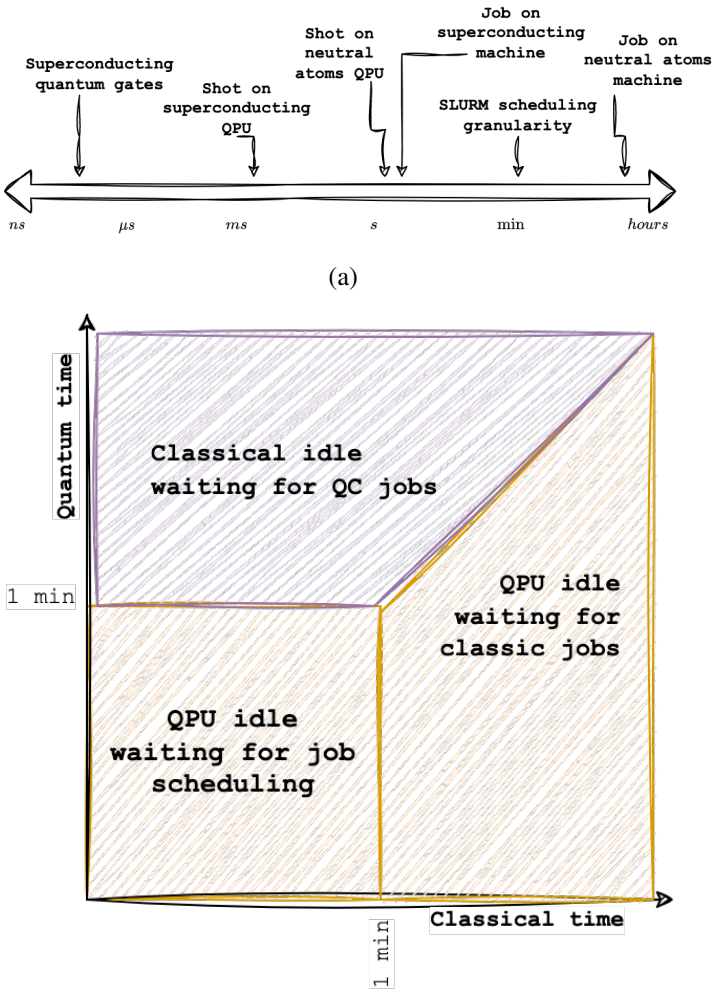


Fig. 5.2 Time scales in hybrid QC–HPC execution (a) and related scheduling pathologies of naive co-allocation in hybrid jobs (b).

scheduling and the related scheduling pathologies in the case of naive co-allocation in hybrid jobs.

If a hybrid job reserves both partitions exclusively for the full wall time, the resulting resource waste is structural: either the QPU is held idle while classical computation proceeds, or classical nodes wait idle for quantum phases to complete. This observation motivates the central premise of this Chapter: in the NISQ era, *system-level execution and scheduling are enabling conditions* for scalable hybrid computing, not secondary implementation details.

Recent system efforts in the broader literature reinforce this view. Proposals for HPC–QC integration and middleware increasingly emphasize unified abstractions for workload management across CPUs/GPUs/QPUs, the need to expose quantum resources to HPC control planes, and the importance of runtime services (e.g., asynchronous submission and backpressure) that mitigate the overheads and contention intrinsic to hybrid loops [120–123]. Within this landscape, the contribution discussed in this Chapter focuses on three complementary strategies that address the dominant scheduling bottleneck: (i) loosely coupled workflows, (ii) quantum resource virtualization via virtual QPUs, and (iii) dynamic resource management through malleability. These strategies define a design space in which the best choice depends on the workload's imbalance regime and on the specific characteristics of the targeted quantum technology.

5.2 Architectural Requirements for Efficient Hybrid Execution

To translate the above motivation into actionable system design, it is useful to state explicit architectural requirements for a production-grade hybrid QC–HPC platform in the NISQ era. These requirements follow from two constraints that dominate current deployments: QPUs are scarce and heterogeneous resources accessed through non-uniform interfaces, and hybrid applications alternate phases with radically different resource needs and time scales. The goal is to maximize throughput and predictability under contention while preserving fairness, without forcing application developers to embed cluster-specific scheduling logic into their code. Before introducing the requirements, it is useful to define a reference architecture, shown in Figure 5.3.

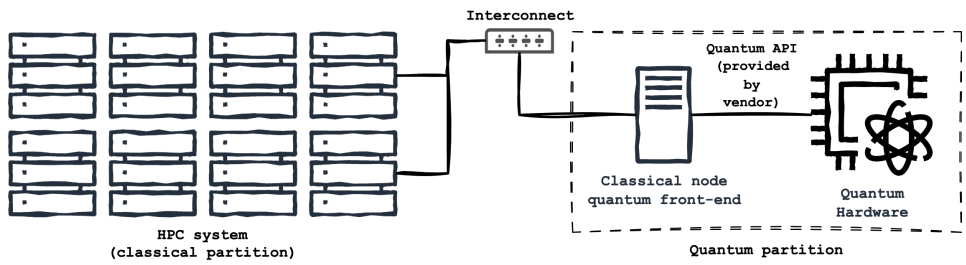


Fig. 5.3 Reference architecture for exposing quantum resources to batch schedulers.

```
#!/bin/bash
#SBATCH --partition classical
#SBATCH --nodes 10
#SBATCH --time=01:00:00
#SBATCH hetjob
#SBATCH --partition quantum
#SBATCH --gres=qpu:1
#SBATCH --time=01:00:00

srun ./hybrid_job
```

Listing 1 Example of SLURM job script to run a potential HPC-QC hybrid job

Indeed, this is the standard modality for exposing heterogeneous resources through batch schedulers. In the specific HPC–QC case, the HPC nodes are defined as `classical partition`, while the `Quantum partition` is composed of the actual QPU and the classical node provided by the vendor (or third-party) which acts as front-end. The front-end usually interacts with the quantum hardware through dedicated (ofter REST) APIs and it is equipped with a queue that is neither directly accessible by nor visible to the batch scheduler. In this setup, schedulers allow users to execute heterogeneous jobs (`hetjob`) using a simple co-scheduling model, referred to as *baseline* throughout the rest of this Chapter. The (SLURM) script takes a form similar to Listing 1:

However, since the scheduler sees only one QPU, we fall back to the sequential execution of hybrid jobs in the multi-tenant environment of HPC clusters, which produces the previously defined pathologies. Hence, the need for solutions that satisfy the following requirements.

5.2.1 R1: Compatibility with operational HPC control planes

Quantum resources should be integrated into the same policy and accounting domain as classical resources. Operational HPC infrastructures enforce priorities, quotas, and fair sharing through batch schedulers (e.g., SLURM), while QPUs are typically accessed through separate vendor queues and accounting systems. Efficient hybrid execution therefore requires a bridging layer capable of exposing QPU capacity to the HPC scheduler in a controlled manner, mapping scheduler-level allocations to device-level submissions, and reconciling accounting across both domains [7, 122]. A minimal mechanism is the representation of QPUs as schedulable entities (e.g.,

via generic resources), but this alone is insufficient if allocation remains static and exclusive, as it reproduces the idle-time pathologies induced by hybrid imbalance.

5.2.2 R2: Support for multiple integration modes

A hybrid platform should support at least two integration modes reflecting the maturity of devices and applications: (i) quantum offloading, where a classical application invokes quantum kernels as accelerators, and (ii) loosely coupled workflows, where an orchestrator schedules classical and quantum steps separately while preserving dependencies [7, 120]. Offloading emphasizes low-latency invocation, asynchronous submission, and efficient multiplexing of short quantum bursts. Workflow-based execution emphasizes dependency management and the ability to allocate resources only when needed, which is crucial when quantum phases are long enough that co-allocation would otherwise waste substantial classical capacity.

5.2.3 R3: Mechanisms to mitigate workload imbalance

Because imbalance is the dominant source of inefficiency, the platform must provide explicit mechanisms that target it. We propose three complementary approaches.

First, *workflow decomposition* mitigates over-allocation by allocating classical and quantum resources independently for each step. It is most effective when individual steps are sufficiently coarse-grained to amortize scheduling and queueing overheads. Its limitation is that, for very fine-grained quantum bursts, the overhead of repeatedly entering the scheduler and vendor queue can dominate.

Second, *quantum resource virtualization* through *virtual QPUs* mitigates QPU underutilization by enabling temporal multiplexing of a single physical QPU among multiple jobs. Applications request vQPUs, while the infrastructure interleaves their quantum submissions on the same device up to a configured multiplexing degree. Architecturally, the core property is transparency: application quantum code should remain unchanged while policies and accounting remain enforceable at the scheduler level.

Third, *malleability* mitigates both CPU and QPU waste by allowing the classical portion of a hybrid job to adapt its allocation at runtime, shrinking during quantum phases and expanding when returning to classical computation. This can reduce idle

time and queue-induced delays in regimes where workflow decomposition would be too overhead-dominated and where virtualization alone yields limited benefit. Its main drawback is implementation complexity, as it requires runtime support (e.g., dynamic process management) and careful application-level coordination.

5.2.4 R4: Separation of concerns between application, runtime, and scheduler

Hybrid efficiency should not be achieved by hard-coding cluster policies into application logic. Instead, concerns should be separated: the application describes its hybrid structure (e.g., quantum-phase boundaries or workflow dependencies), a runtime/middleware layer implements submission strategies (batching, asynchronous execution, backpressure), and the scheduler enforces global policies and allocates resources [120, 121, 123]. This separation enables portability across sites and devices, while retaining the ability to optimize end-to-end performance in a platform-aware manner.

5.2.5 R5: Observability, predictability, and fairness under contention

Finally, efficient hybrid execution must remain stable under contention. Since QPUs are scarce, multi-tenant access is the default, and the platform should provide observability (queue depth, service times, effective throughput), predictability (bounded variance of time-to-solution where possible), and fairness (preventing starvation and enforcing quotas) [122]. This becomes especially relevant when adopting virtualization: although vQPUs can increase device utilization, they may introduce per-job latency due to time sharing. The platform should therefore make the latency–throughput trade-off explicit and controllable via policy (e.g., limits on multiplexing, class-based prioritization, or reservations for latency-sensitive workloads).

The remainder of Chapter 5 instantiates these requirements into concrete architectural solutions. In particular, the next sections introduce quantum resource virtualization through the vQPU abstraction and its integration within an HPC scheduling model, followed by quantitative characterization of the resulting trade-

offs and by complementary approaches such as workflow-based decomposition and malleable execution.

5.3 Quantum Resource Virtualization: The vQPU Approach

The vQPU approach addresses a practical mismatch between how current HPC batch schedulers account for quantum accelerators and how many hybrid workloads actually use them. In the baseline allocation model, a hybrid job requests classical resources together with exclusive access to the QPU for the full wall time of the job. Since the quantum phase is typically interleaved with non-negligible classical phases (pre/post-processing, parameter updates, orchestration, aggregation), this policy easily leads to long periods in which the QPU is *reserved but idle*. In multi-user environments, this becomes a system-level inefficiency because other jobs are prevented from starting even their classical computation while the QPU sits unused.

The key idea of vQPUs is to virtualize the quantum resource at the scheduler level by turning a single physical QPU into multiple *time-shared* scheduler-visible tokens. Each token represents a *share of access over time*, not a spatial partition of qubits. As a consequence, multiple hybrid jobs can be admitted concurrently by the HPC scheduler, while the physical device continues to execute quantum submissions sequentially, typically through a vendor-provided internal FIFO queue. In other words, vQPUs enable *concurrent classical execution* and *serialized quantum execution*, with the serialization delegated to the QPU control plane and its queueing mechanism.

This conceptual shift is illustrated by contrasting the baseline policy (Figure 5.1 with the vQPU policy (Figure 5.4). In the baseline timeline, Job 1 holds the only QPU allocation for its entire duration, thereby forcing Job 2 to wait even before beginning its classical phase; the resulting idle gaps on the QPU translate directly into avoidable cluster-level wall time. Under vQPUs, instead, Job 1 and Job 2 are both admitted (each holding one vQPU token) and can execute their classical segments in parallel. Whenever either job reaches a quantum segment, it submits circuits to the physical QPU, which serves them sequentially via its FIFO queue. The price paid for this increased concurrency is an additional *quantum queue overhead*

in each job runtime, caused by waiting for the device; however, when the workload is sufficiently imbalanced (classical phases much longer than quantum phases), the overhead is typically small compared to the global throughput gains.

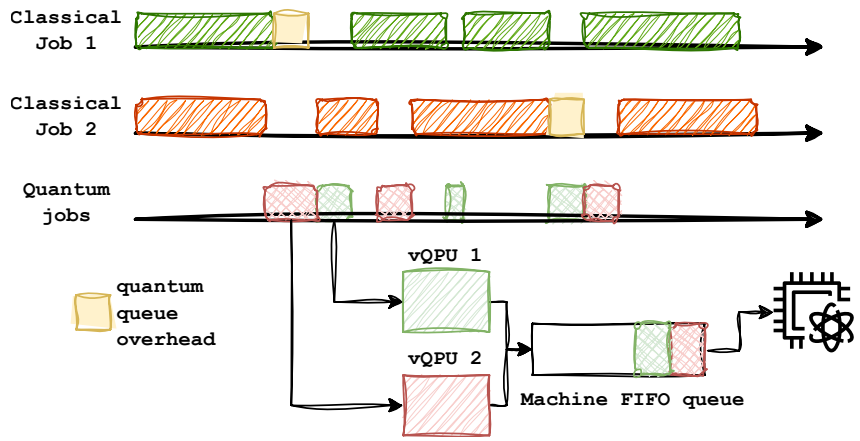


Fig. 5.4 Time-based multiplexing with vQPUs: both jobs start concurrently; quantum phases are serialized by the device FIFO, inducing a queue overhead that is amortized when classical and quantum times are imbalanced.

From a systems perspective, the vQPU abstraction can be described by three elements that are useful to make explicit in the thesis narrative.

First, vQPUs are an *admission-control mechanism* at the HPC scheduler boundary. The scheduler grants up to n_{copies} concurrent hybrid allocations targeting the same QPU (the multiplexing degree), thus preventing the pathological sequentialization induced by exclusive QPU booking. The scheduler is therefore responsible for deciding *how many* jobs may overlap in time, while the physical QPU remains a single-server resource.

Second, the QPU front-end remains the *service-order mechanism*. Circuit submissions from all concurrently running jobs flow into the device-side FIFO queue and are executed one after the other. This preserves compatibility with the operational constraints of present-day gate-based QPUs, where genuine hardware-level parallelism across independent users is generally not available, and multi-tenancy must be realized through time multiplexing and queueing.

Third, vQPUs make the latency–throughput trade-off *explicit and tunable*. Increasing n_{copies} tends to increase quantum occupancy (reducing idle time), but also increases waiting times at the quantum queue once contention becomes significant. Hence, vQPUs are most beneficial in the regime where quantum work arrives in

short bursts and can be efficiently interleaved across many jobs, while abundant classical resources can sustain concurrent classical phases.

In the experimental study detailed in Section 5.4, the vQPU mechanism is evaluated through a controlled workload based on multiple concurrent replicas of the same hybrid application. In particular, we decided to implement the BBQ-mIS case-study described in Section 4.5, using a digital version of QAOA for the quantum subroutine. The quantum part solves a MIS subproblem through QAOA (with a fixed circuit structure per instance), while the classical part performs the algorithmic orchestration and associated processing steps.

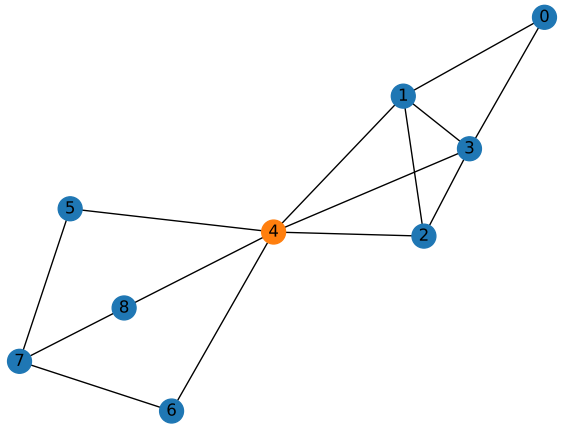
To enable experiments that exceed the qubit capacity of the available hardware, we relied on a circuit cutting procedure [124] that decomposes a larger circuit into smaller sub-circuits executed separately and recombined classically. Since circuit cutting is a hardware-driven enabling technique and has not been introduced previously in this thesis, in the following we only retain the minimum systems-relevant intuition: circuit cutting modifies the *effective duration and granularity* of quantum phases, and therefore directly affects queueing dynamics under time multiplexing. A concrete example is shown in Figure 5.5, where a QAOA circuit is split into two smaller circuits by identifying a cut location (the "cut-move" reference gate), without requiring additional ancilla qubits at execution time.

5.4 Performance Characterization of vQPU

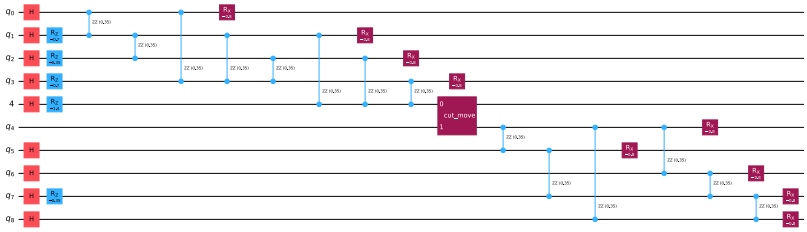
This section summarizes the performance characterization of vQPUs, focusing on cluster-level metrics that quantify (i) the degree to which time multiplexing increases utilization of a scarce QPU and (ii) the extent to which such gains translate into reduced overall turnaround time when multiple hybrid jobs are concurrently executed.

5.4.1 Experimental setup and workload model

The experimental design aims to reproduce a stress scenario in which multiple concurrent hybrid workloads compete for the same QPU. To isolate scheduling effects, the study launches n_{copies} concurrent replicas of the same hybrid application, each executing the same problem instance but subject to intrinsic probabilistic



(a) A (1,5)-clustered graph used as input to the GC problem. The central node (label 4) is the vertex separator of 2 clusters with 5 nodes each.



(b) The QAOA circuit used to solve a MIS instance for the graph in (a). The "cut-move" gate in central position serves as a reference for the circuit cutting operation, thereby dividing the whole 9-qubits circuit into two 5-qubits' ones. Note that the circuit cutting operation introduces an additional wire into the circuit, but at the execution stage no ancilla qubit is involved.

Fig. 5.5 Example of a (1,5)-clustered graph used as BBQ-mIS input (a) and the corresponding 9-qubit QAOA circuit with an identified circuit-cutting gate (b). The cut-move gate divides the circuit into two 5-qubit sub-circuits executed separately and recombined classically, without requiring ancilla qubits at execution time.

variability due to quantum sampling. The baseline configuration corresponds to exclusive allocation of the QPU to each replica, forcing sequential processing of jobs. The vQPU configuration, instead, admits concurrent execution of all replicas (up to the chosen n_{copies}), while the quantum submissions are serialized through the device FIFO.

A key parameter is the *quantum–classical workload ratio*, denoted R, which controls the imbalance between classical and quantum durations. Intuitively, R models how much classical computation is present between (or around) quantum submissions. Since vQPUs rely on overlapping classical work of some jobs with quantum work of others, increasing the imbalance (larger classical components) should increase the opportunity for beneficial overlap and therefore amplify the advantage of time multiplexing.

In order to correctly evaluate the metrics defined in Section 5.4.2, the experiment required the execution of a batch job in a production HPC environment with access to both classical and quantum resources. Therefore, the classical workload was carried out on the Leonardo supercomputer, hosted at the CINECA facilities, while the quantum workload was dispatched to the "Lagrange" quantum computer[2], a gate-based, 5-qubit Spark-model device by IQM, hosted at Politecnico di Torino and acquired and installed in the context of the *L-QCS initiative*. The QPU was accessed using the standard IQM API in a cloud-like environtment, through a custom middleware whose design is part of the author's contribution [4]. However, the machine was reserved for the experiment in order to avoid unwanted queueing times and latencies. This does not affect the results since the load is controlled by varying the number of concurrent workloads as mentioned above. The resource scheduler used for the experiment is SLURM. The experimental workflow is organized around a main loop in which, at each iteration, an increasing number of GC instances is submitted simultaneously. This setup was designed to emulate concurrent access to the QPU by multiple users.

The experimental design assumes that the number of available vQPUs always matches the number of concurrent GC instances. This assumption is motivated by methodological considerations: if fewer vQPUs than submitted jobs were available, the scheduler would keep the excess jobs on hold until both quantum and classical resources were available, as it normally does. Conversely, allocating more vQPUs

[2]https://pages.linksfoundation.com/links-iqm-spark/docs/

than the number of concurrent jobs would not alter the execution in any meaningful way. Ultimately, both cases do not provide additional insight into the experiment.

5.4.2 Metrics

Several complementary metrics can be defined as functions of n_{copies} (the concurrency level) and R (the workload imbalance).

The first metric is the *total time* required to complete all n_{copies} replicas. This cluster-level quantity captures throughput: a lower total time indicates that more hybrid work is completed per unit time.

The second metric is the *quantum occupancy*, defined as the fraction of time in which the physical QPU is actively executing circuits (as opposed to being idle). This metric directly quantifies whether the multiplexing policy is effective at reducing idle gaps on the QPU.

The third metric is the *quantum time*, i.e., the cumulative time actually spent executing circuits on the QPU (excluding queueing delays). This value is expected to scale approximately linearly with n_{copies} and to be largely independent of R, since R affects classical work rather than the amount of quantum computation submitted.

Finally, the study reports the *mean queue time*, defined as the average cumulative waiting time experienced by a job before gaining access to the quantum computer for circuit execution. This metric captures the principal cost of multiplexing: as more jobs compete for the same device, queueing delays increase, potentially reducing the per-job benefit.

These metrics are presented in Figure 5.6 (quantum occupancy, quantum time, mean queue time) and Figure 5.7 (total time and runtime distributions), and are averaged over multiple runs with standard deviations indicated.

5.4.3 Observed trends and interpretation

A first central observation is that vQPUs significantly reduce the overall completion time compared to the baseline, and that this advantage increases with workload imbalance. The total time curves decrease markedly when moving from exclusive allocation to time multiplexing, and the gap between vQPU and baseline widens for

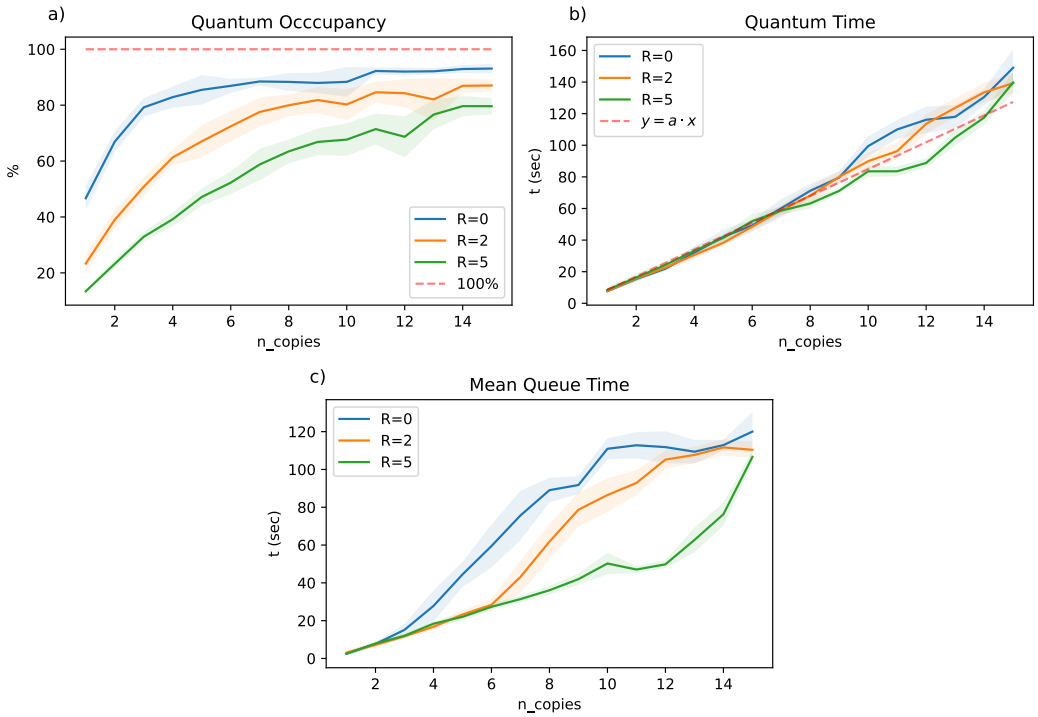


Fig. 5.6 Quantum occupancy, quantum time, and mean queue time as functions of the number of concurrent replicas n_{copies}, for different quantum–classical workload ratios R.

larger values of R. This is consistent with the mechanism of overlap: as classical segments become longer relative to quantum segments, the scheduler can keep more jobs progressing on classical resources while the QPU is continuously fed with back-to-back quantum submissions from different jobs, thereby reducing idle time and increasing throughput.

The quantum occupancy curves further support this interpretation. As n_{copies} increases, occupancy increases up to a saturation region in which the QPU becomes nearly fully utilized. Before saturation, adding more concurrent replicas primarily fills previously idle gaps, improving both occupancy and total time. After saturation, however, the QPU behaves as a continuously busy single-server resource and additional concurrency can no longer increase occupancy; instead, it increases contention, pushing more time into queueing.

This transition is reflected by the mean queue time behavior. For small n_{copies}, queueing remains limited because the QPU is not always saturated and jobs frequently find the device available upon request. As n_{copies} grows, mean queue time

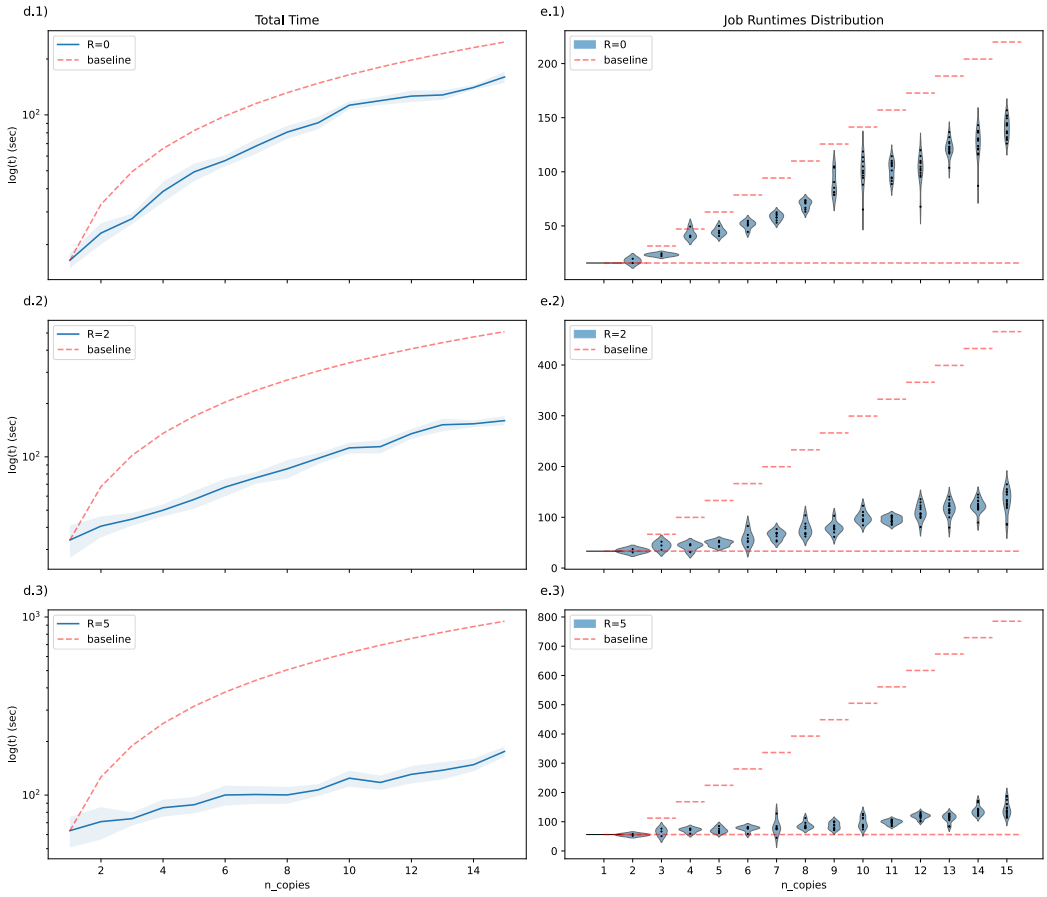


Fig. 5.7 Total completion time and per-job runtime distributions under vQPUs, compared to the baseline policy, for varying n_{copies} and R.

increases because more jobs compete for access, and the curves show an upward trend that becomes more pronounced near saturation. Importantly, the impact of this queueing cost depends on R: for larger workload imbalance, the relative burden of queueing is reduced because classical phases provide more opportunity to "hide" waiting behind useful computation, thus preserving the net gain of multiplexing.

The quantum time metric behaves as expected: it scales linearly with n_{copies} and overlaps across different values of R, because it measures only the device execution time (excluding waiting) and is therefore insensitive to classical workload variations. This separation between quantum time and queue time is conceptually important for system design: improvements due to vQPUs do not come from reducing the intrinsic quantum cost of the workload, but from increasing the *duty cycle* of the QPU by reducing idle periods.

While the cluster-level throughput improves, the per-job runtime distributions reveal a more nuanced user-level effect. The violin plots reported in Figure 5.7 show that, under worst-case simultaneous submission, some jobs may experience more pronounced delays relative to their baseline counterparts due to queueing. However, the distribution as a whole shifts toward shorter completion times for the majority of concurrent jobs, and the cluster-level total time decreases accordingly. This captures the fundamental trade-off of time multiplexing in a single-server accelerator: increased global efficiency may come at the expense of longer runtimes for a subset of jobs, particularly those that suffer from unfavorable queue positions during periods of saturation.

Overall, the performance characterization supports the view that vQPUs are most effective for hybrid workloads with pronounced quantum–classical imbalance and bursty quantum phases, where multiplexing can substantially increase QPU occupancy and reduce cluster-level completion time. At the same time, the results make explicit that the multiplexing degree n_{copies} must be chosen carefully: once the QPU is saturated, further increasing concurrency primarily increases queueing delays, reducing the marginal benefit and potentially worsening tail latencies. These insights motivate the need for policy-level controls (e.g., limiting n_{copies} per queue or per user class) and for runtime observability of queueing.

5.5 Malleability and Workflows for Hybrid QC–HPC Applications

Hybrid QC–HPC applications rarely exhibit a stationary resource footprint. In most practical NISQ-era pipelines, classical phases (e.g., data preprocessing, orchestration, post-processing, and often the dominant part of the optimization loop) alternate with quantum phases (e.g., sampling, expectation estimation, or the solution of a sub-problem formulated as QUBO/Ising). This alternation creates a structural mismatch with the static allocation model that dominates HPC production systems, where a batch scheduler grants a fixed set of resources to a job for its entire wall time. In the QC–HPC setting, such staticity translates into systematic waste: classical nodes can be held idle while the job is waiting for, or executing on, a scarce quantum resource; conversely, reserving only classical resources and offloading the quan-

tum part through a "cloud-like" API introduces the opposite inefficiency, since the application stalls on the classical side while waiting for the quantum service, still retaining its classical allocation.

Two complementary strategies address this issue by restoring *dynamicity* at the system level while preserving compatibility with established HPC control planes: (i) a *workflow-based decomposition* of the application into schedulable steps; and (ii) a *malleability-based* approach where a single job can reshape its allocation during runtime. Both approaches are summarized here with a focus on their programming and scheduling implications.

5.5.1 Workflow-based resource allocation

A workflow approach represents the application as a set of tasks connected by explicit dependencies, and delegates to a Workflow Management System (WMS) the orchestration of task execution across heterogeneous resources [125, 126]. Conceptually, a hybrid workflow can be expressed as (W, L, M), where $W = (S, P, D)$ encodes the workflow graph with steps S, ports P, and dependency links D, L is the set of available execution locations, and $M \subseteq (S \times L)$ maps each step to one or more feasible locations. This abstraction is convenient in QC–HPC because it makes the heterogeneity explicit: classical steps map to CPU/GPU partitions, while quantum steps map to a QPU partition or to a service endpoint exposed through a connector [119, 127]. Figure 5.8 provide a visual example of the coupling between the various steps of a workload and the execution environment.

A practical advantage of the workflow decomposition is that it enables *separate resource reservations* for classical and quantum phases. The scheduler allocates classical nodes only for the steps that actually use them, and releases them while the workflow transitions to a quantum step. In other words, the workflow naturally matches the temporal structure of hybrid applications, mitigating the "hold-and-wait" pathology typical of monolithic jobs.

In this context, StreamFlow provides a representative implementation [128]. StreamFlow builds on the Common Workflow Language (CWL) [129] and enriches it with (i) a topology of deployment locations and (ii) a loop construct to encode iterative workflows. This is particularly relevant for NISQ workloads, where the loop structure (e.g., training epochs, variational iterations, or repeated solver calls)

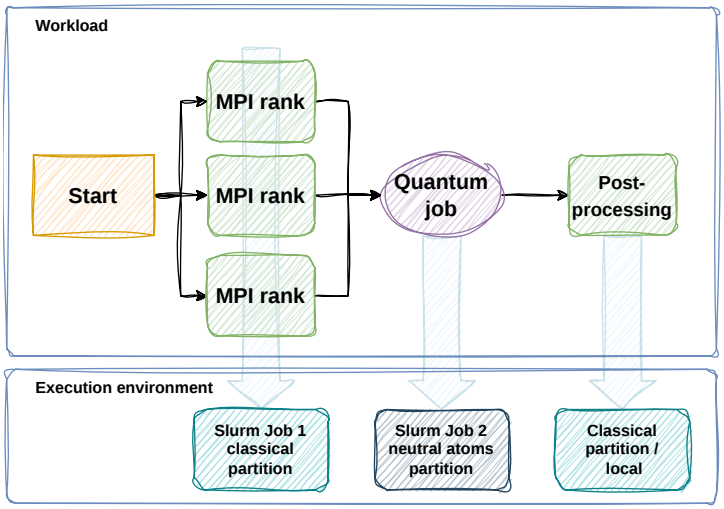


Fig. 5.8 A Quantum-HPC workload executed as a loosely-coupled, independently scheduled workflow.

is a first-order driver of both performance and resource contention. StreamFlow also adopts a connector model: integrating a new quantum backend reduces to implementing a connector that mediates either a REST-like API or a queue-based interface to a SLURM-managed quantum partition.

From a software-engineering viewpoint, workflows shift complexity from *in-process* state management to *inter-step* orchestration: a monolithic hybrid code is refactored into modular components with clear inputs and outputs, which are then "glued" by the workflow definition. This refactoring cost is the main practical barrier to adoption, but the resulting separation of concerns is often aligned with the requirements discussed earlier in this Chapter (notably, clearer boundaries between application logic, runtime orchestration, and scheduler-facing resource requests).

5.5.2 Malleability-based resource allocation

Malleability addresses the same inefficiency, i.e., static resource allocation under time-varying requirements, but does so without necessarily decomposing the program into multiple externally scheduled steps. A malleable application can grow or shrink the amount of allocated resources at runtime, reacting to the application phase and, potentially, to cluster conditions. In a QC–HPC workload, the key opportunity is

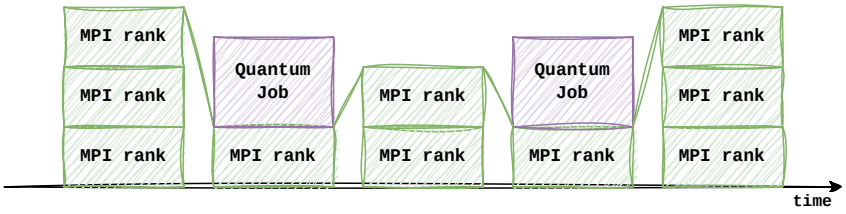


Fig. 5.9 Representation of a workload execution featuring malleability properties to handle the release and retrieval of resources when switching from classical to quantum-oriented workloads.

to *release classical resources during quantum phases* and reacquire them when computation returns to the classical part, thereby preventing idle reservations while still preserving a single logical job. Figure 5.9 provides a schematic of malleable execution of a workload in an hybrid cluster.

A representative toolchain is DMR, a framework that supports passive dynamic resource management through a set of synchronization points embedded in the application [130]. The high-level behavior is as follows. During execution, the application reaches predefined reconfiguration points. At these points, DMR negotiates with the resource manager (e.g., SLURM through an integration layer) to determine whether a reshape is feasible, coordinates process-level changes through the parallel runtime, and then resumes execution with the new "job shape" (i.e., the new number of nodes/processes). In this model, the burden shifts to application state management: the code must expose a checkpoint/restart logic sufficient to reconstitute progress after a reshape.

A particularly natural mapping for hybrid QC–HPC loops is to treat each iteration as consisting of a classical stage and a quantum stage. The malleable code then expands (or ensures sufficient resources) before entering the classical stage, and minimizes the allocation before entering the quantum stage, potentially shrinking down to a single rank that remains active only to offload the quantum computation and to drive the subsequent restart [130]. A schematic of this structure is given in Listing 2, adapted from a reference implementation.

Relative to workflows, malleability offers a different trade-off. It can be integrated into an existing MPI codebase with limited refactoring, but it requires more careful reasoning about program state, checkpoint granularity, and correctness under reshapes. From a platform viewpoint, malleability also relies on tighter coupling with

```
void main() {
  MPI_Init();
  DMR_INIT (restart(it0, quantum));
  for (auto it = it0; it < ITERS; it++) {
    if (!quantum) {
      while (!resources && !timeout)
        DMR_EXPAND (checkpoint(it, quantum));
      //Execute Classical Computation
      quantum = true;
      DMR_MINIMIZE (checkpoint(it, quantum));
    }
    if (rank_id == 0)
      //Offload Quantum Computation
    quantum = false
  }
  MPI_Finalize();
}
```

Listing 2 Scheme of how malleability is adopted in the user code.

the scheduler and the parallel runtime, which can be advantageous in HPC-centric deployments but may reduce portability across execution environments compared to a workflow system with multiple backend connectors.

5.5.3 Use-case driver: clustering aggregation as a hybrid QC–HPC workload

To ground the discussion of workflows and malleability in a concrete execution pattern, a hybrid application for *clustering aggregation* is considered. The goal is to obtain a robust consensus clustering by combining the outputs of multiple clustering algorithms [131]. This use case is representative for two reasons: the classical portion is naturally parallel (multiple clustering methods can run concurrently), and the quantum (or quantum-emulated) portion is invoked only intermittently to solve a combinatorial aggregation problem.

Given a dataset, three common clustering methods are executed: k-means [132], DBSCAN [133], and hierarchical clustering [134]. In the HPC-adapted version, each method is assigned to an MPI rank (or, more generally, to a distinct node), so that the three clusterings are computed in parallel. The three results are then merged by building an undirected *overlap graph* whose nodes represent candidate clusters and whose edges represent overlap (shared data points) between clusters produced by

different algorithms. In this formulation, a valid consensus clustering corresponds to an Independent Set of the overlap graph, because clusters in the final solution must be non-overlapping. Selecting a maximum(-weight) Independent Set therefore yields a consistent aggregation of the individual clusterings [131].

The Independent-Set selection is written as a QUBO objective. Using the notation of the reference implementation, let $x_i \in \{0,1\}$ indicate whether cluster i is selected, let w_i be a weight (chosen to promote balanced cluster sizes), and let λ be a penalty that discourages selecting overlapping clusters. Similarly to Equation 4.6, the objective can be expressed as

$$f_{\mathrm{MIS}}(x) = -\sum_{i \in V} w_i x_i + \lambda \sum_{(i,j) \in E} x_i x_j, \tag{5.1}$$

which directly matches the QUBO form. The QUBO is then passed to a solver suitable for the available quantum technology. The reference study solves it via simulated annealing on a dedicated "quantum" node, as a proxy for quantum annealing-like behavior [135]; however, the same interface can target analog quantum processors (e.g., neutral atoms, quantum annealers) or gate-based QPUs (e.g., via QAOA) [136].

Finally, the application evaluates the clustering quality via the silhouette score and iterates until a target threshold is reached (or a maximum number of iterations is exceeded). This iterative structure makes the use case particularly suitable for studying dynamic allocation: the workflow/malleability logic can be exercised multiple times, with repeated transitions between a parallel classical phase and a short (or, depending on technology, potentially long) quantum phase. A schematic of the hybrid pipeline is provided in Figure 5.10.

5.6 Performance Characterization of Dynamic Solutions

This section characterizes the system-level behavior of the two dynamic allocation strategies introduced in Section 5.5 when applied to the clustering aggregation workload of Section 5.5.3. The goal is not to optimize the clustering algorithm itself, but to quantify how different allocation models impact time-to-solution and

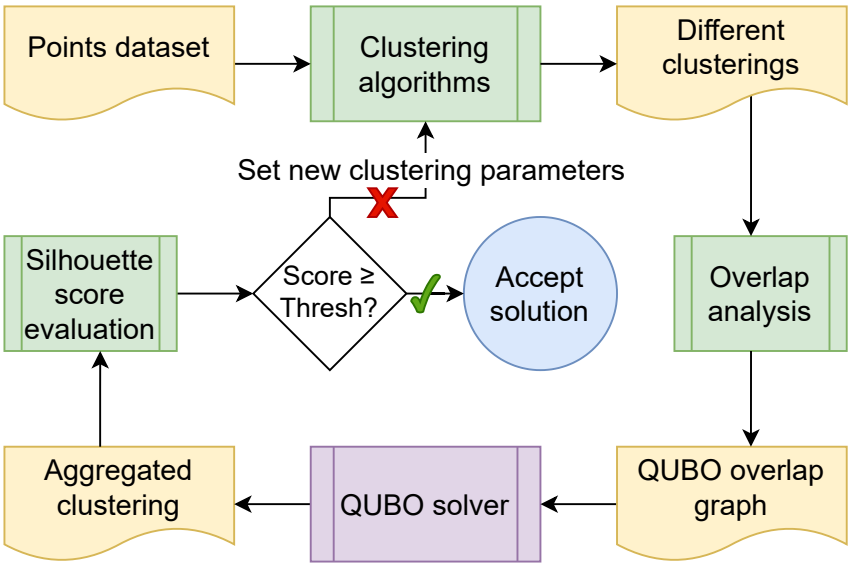


Fig. 5.10 Hybrid clustering aggregation pipeline. The workflow alternates parallel classical clustering steps with a QUBO/MIS aggregation step suited for quantum or quantum-emulated resources, and a silhouette-based evaluation step driving the loop termination.

classical resource utilization under representative hybrid execution patterns. In contrast to the vQPU study of Section 5.4, which dispatched quantum circuits to a real QPU (the Lagrange IQM Spark device), the experiments here use an emulated quantum subroutine: a Simulated Annealing solver runs on a dedicated classical node as a proxy for quantum annealing-like execution. This is not a limitation for the strategies under study, which are agnostic to how the quantum computation is performed; what matters is the temporal structure of the hybrid workload, which the emulation preserves. The results summarized below follow the experimental campaign presented in the reference article.

5.6.1 Experimental setup and execution scenarios

The reference evaluation is performed on a small SLURM-managed cluster configured to emulate a plausible heterogeneous QC–HPC environment. The infrastructure includes a classical partition (three CPU-only nodes) used to execute the parallel clustering methods, and a separate “quantum” partition (one node) used to run the QUBO solver and to emulate a QPU. The workflow-based approach uses StreamFlow to allocate resources step-by-step, while the malleability-based approach uses DMR

to reshape the allocation within a single job. In both cases, the quantum phase is dispatched to the quantum partition through a queue-based mechanism.

To isolate the impact of allocation, three execution configurations are compared.

Baseline (static co-allocation). A single SLURM job co-allocates the classical nodes for the full duration of the application. The quantum subtask is offloaded to the quantum partition through an external submission mechanism, while the classical nodes remain reserved even when not computing.

Workflow-based dynamic allocation. The application is decomposed into workflow steps and executed by the WMS. Classical nodes are allocated for the clustering and evaluation steps, while the quantum node is allocated only when entering the QUBO step. The loop is represented explicitly in the workflow through an iterative construct.

Malleability-based dynamic allocation. The application is submitted as one logical job but shrinks before the quantum phase, releasing most classical resources, and expands again when returning to the classical phase. In the simplest form, the job minimizes down to a single MPI rank that remains active to trigger the quantum offload and to re-expand afterwards.

5.6.2 Metrics

Two metrics are used to capture complementary aspects of performance. The first is the end-to-end wall time, including scheduler overheads for job initialization/finalization (and, for workflows, repeated scheduling of steps). The second is classical resource usage expressed in node-seconds, i.e., the integral over time of the number of allocated classical nodes. This second metric quantifies resource waste directly, and is particularly meaningful in hybrid settings where the quantum resource is scarce but the classical allocation can be large and expensive.

5.6.3 Workload model and emulated quantum time scales

A key observation in QC–HPC is that the relative duration of the quantum phase is technology dependent. Gate-based superconducting QPUs typically execute jobs on a short time scale (order of seconds, dominated by orchestration and repeated circuit execution), whereas analog platforms such as neutral atoms may require longer times (order of minutes when including preparation and execution). To reflect this variability while keeping the pipeline unchanged, the reference study profiles two versions of the same workload: one where the quantum phase completes in less than a second (proxy for fast QPU execution), and one where an artificial delay of minutes is added during the quantum phase (proxy for slower QPU execution). The dataset used in the clustering application is generated via `make_blobs` and is sized so that the iterative loop terminates after a fixed number of iterations once the silhouette threshold is exceeded, enabling reproducible comparisons.

5.6.4 Observed trends and results

The main trend emerging from the experiments is that both dynamic approaches reduce classical resource usage substantially relative to the baseline, with different time-to-solution trade-offs. Table 5.1 reports measured wall-time for 2 minutes long quantum jobs in the case of a single workload and two concurrent ones. The results show that when the quantum phase is long (minutes-scale), releasing classical nodes during the offload eliminates large idle intervals, so node-seconds drop sharply for both workflow and malleability. In this regime, workflows often achieve the lowest classical consumption because classical resources can be fully released between steps, while malleability typically retains at least one active process to preserve the single-job semantics and enable quick resumption after the quantum phase.

Wall time behavior depends on where overhead is paid. The workflow strategy may incur additional latency from repeatedly requesting resources at each step and from the orchestration overhead of the WMS, especially in iterative pipelines. Malleability, by keeping execution within one job and reshaping at synchronization points, can reduce such repeated scheduling overheads while still avoiding most classical waste. In the reported experiments, malleability therefore acts as a compromise: it captures most of the resource savings of workflows while remaining closer to baseline in time-to-solution.

Table 5.1 Comparison between Baseline, Workflow and Malleability execution types for hybrid workloads with 2 minutes quantum jobs. The table shows results for a single workload (Single Mode), and two concurrent copies (Double Mode).

Execution Type	Mode	Wall time [seconds]	Resource usage [node-seconds]
Baseline	Single	1019.58 ± 0.85	3058.74 ± 2.56
Workflow	Single	1057.80 ± 6.02	1161.20 ± 6.94
Malleability	Single	1029.06 ± 1.54	1647.75 ± 1.54
Baseline	Double	2038.43 ± 0.96	6115.30 ± 2.89
Workflow	Double	1226.00 ± 1.58	2324.00 ± 3.39
Malleability	Double	1127.65 ± 1.18	2817.73 ± 1.27

Under higher concurrency (multiple simultaneous workloads), dynamic strategies provide an additional advantage: by relinquishing classical nodes during quantum phases, they allow different workloads to interleave their classical segments more effectively, improving overall throughput. In contrast, the baseline strategy tends to serialize progress because classical allocations remain occupied even when the job is blocked on the quantum phase, reducing the scheduler's ability to start other work. This effect is particularly visible when the quantum phase is long enough to create sizeable "waiting windows" that can be exploited by the scheduler, as shown in Figure 5.11.

Indeed, in the case of quantum jobs with duration less than a second, the benefit in terms of wall time decreases significantly, especially for the workflow approach, as shown in Table 5.2. However, the different performance with respect to malleability comes from the intrinsic properties of the algorithms executed during the classical phase (with *k*-means taking significantly less time than the other two), so they do not represent a trend. In general, scenarios involving short quantum jobs and no resource contention should show limited resource savings when using either workflow or malleability approaches.

The above characterization reinforces the general requirements discussed earlier in this Chapter, especially the need for mechanisms that mitigate workload imbalance between abundant classical resources and scarce quantum resources. Workflows and malleability address the same bottleneck at different layers of the stack. Workflows externalize the control flow and make phase boundaries explicit to the scheduler, which is ideal for releasing resources completely but introduces orchestration overhead and refactoring costs. Malleability internalizes phase transitions within the

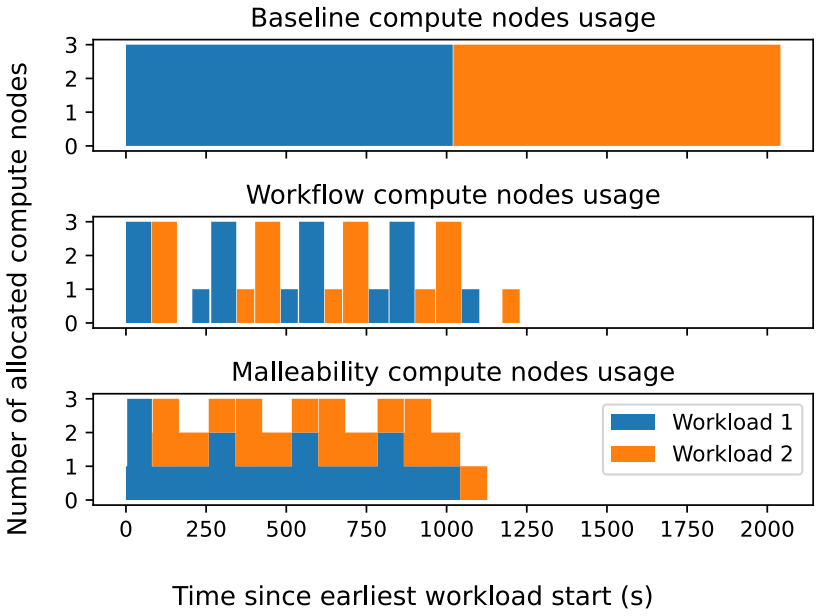


Fig. 5.11 Timeline of cumulative classical node usage for two concurrent workloads under baseline, workflow, and malleability. The plot highlights how dynamic strategies reduce idle reservations and enable interleaving of classical phases.

application, which can preserve a monolithic programming model and reduce repeated scheduling overhead, but requires careful state management and depends on tighter runtime–scheduler integration.

These observations also clarify why dynamic allocation strategies should be evaluated jointly with the expected quantum time scale of the target hardware. If quantum phases are extremely short and contention is negligible, the opportunity for reclaiming classical nodes is limited and the benefits may be dominated by overheads. Conversely, as soon as quantum phases become non-negligible (or contention increases), both workflow-based and malleability-based solutions can yield pronounced improvements in cluster efficiency, creating a direct systems-level incentive to adopt dynamic execution models in production-grade QC–HPC environments.

Table 5.2 Comparison between Baseline, Workflow and Malleability execution types for hybrid workloads with short ($<$ 1 second) quantum jobs. The table shows results for a single workload (Single Mode), and two concurrent copies (Double Mode).

Execution Type	Mode	Wall time [seconds]	Resource usage [node-seconds]
Baseline	Single	539.44 ± 0.53	1618.33 ± 1.60
Workflow	Single	569.00 ± 3.94	1148.00 ± 1.87
Malleability	Single	549.60 ± 1.86	1168.29 ± 1.81
Baseline	Double	1076.98 ± 1.79	3230.95 ± 5.37
Workflow	Double	1089.00 ± 1.00	2324.00 ± 4.24
Malleability	Double	648.61 ± 2.08	1622.63 ± 1.05

5.7 Comparative Analysis of Execution Strategies

The analyses presented in the previous Sections show that there is no universally best execution strategy for hybrid QC–HPC workloads. Rather, the appropriate choice depends on which inefficiency dominates the target scenario, on how explicitly the application can expose its hybrid structure, and on the level of the software stack at which one is willing to intervene. This observation is consistent with the cross-layer perspective of this thesis: inefficiencies in hybrid execution do not arise from a single source, but from the interaction between workload structure, scheduler semantics, runtime behavior, and the scarcity of quantum resources. As a consequence, virtualization, workflow decomposition, and malleability should not be read as competing one-size-fits-all solutions, but as different responses to different manifestations of the same systems-level problem.

A useful way to interpret the design space is to distinguish the three strategies according to the layer at which they operate. The vQPU approach acts at the infrastructure and scheduler interface level. Its objective is to mitigate the under-utilization of a scarce QPU by turning exclusive physical access into time-shared logical access, thereby allowing multiple hybrid jobs to progress concurrently on the classical side while quantum submissions are serialized through the device queue. Workflow decomposition operates at the orchestration level. It makes the hybrid structure explicit by splitting the application into separately schedulable stages, so that classical resources can be allocated only when required by the current step. Malleability acts at the scheduler/runtime boundary. It preserves the abstraction of a single logical application, but allows that application to resize its classical allocation

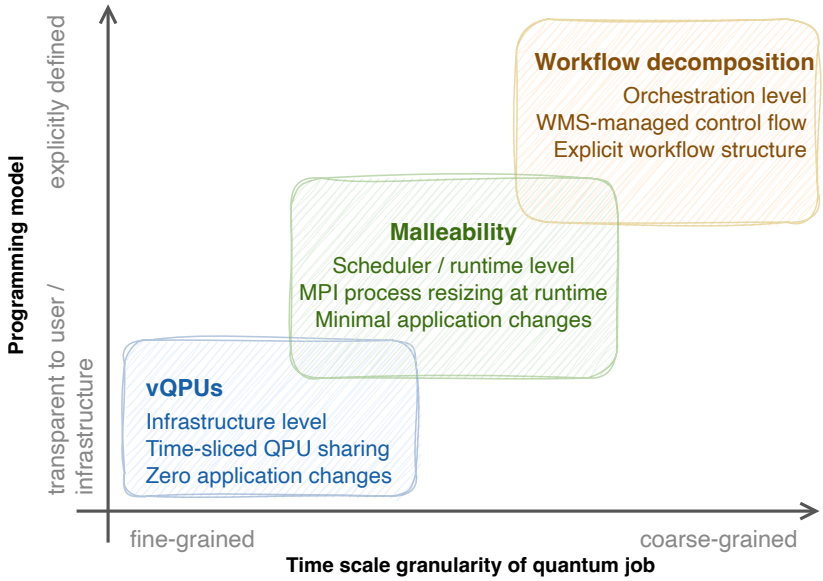


Fig. 5.12 Conceptual positioning of the main execution strategies for hybrid QC–HPC workloads. The figure compares vQPUs, workflow decomposition, and malleability along two dimensions: the time-scale granularity at which the quantum phase is managed and the degree to which the programming model is transparent to the user or explicitly exposed at application level.

at synchronization points, releasing resources during quantum phases and reacquiring them when the execution returns to the classical part. Figure 5.12 summarizes this stratification of responsibilities across the stack.

From a performance standpoint, the three approaches target different primary bottlenecks. The vQPU strategy is most effective when the main problem is poor quantum-device occupancy caused by short, intermittent quantum phases embedded in longer classical computations. In that regime, time-based multiplexing improves cluster-level throughput and reduces the waste associated with reserving a QPU for the full duration of a hybrid job, all without requiring modifications to the user application. However, vQPUs do not directly address the symmetric problem on the classical side: if a job keeps its classical allocation while waiting on a queued quantum request, the QPU is used more efficiently, but some degree of classical inefficiency may remain. In other words, vQPUs primarily optimize the scarce quantum resource and do so through transparency, not through application restructuring.

Workflow decomposition addresses a different form of waste. By making the execution graph explicit, it allows the scheduler to reason in terms of distinct phases rather than monolithic jobs. This is especially attractive when the hybrid application is naturally expressible as a pipeline or iterative DAG, and when the quantum phases are coarse enough that the cost of repeated orchestration remains acceptable. In the experiments discussed in Section 5.6, the workflow strategy achieved the largest savings in classical resource usage, precisely because it avoids holding nodes across phases in which they are not needed. The price paid for this sharper resource matching is additional orchestration overhead, together with the software-engineering effort required to modularize the application and expose its dependencies through a workflow description. The workflow solution is therefore attractive when resource efficiency and portability across heterogeneous backends are more important than minimizing orchestration latency.

Malleability occupies an intermediate position. Like workflows, it addresses the mismatch between static allocation and time-varying resource requirements, but it does so without decomposing the application into externally scheduled steps. Instead, it relies on coordinated reshaping at runtime. This makes malleability particularly appealing for existing MPI-oriented applications, where preserving a single execution context may be preferable to introducing a workflow engine. At the same time, malleability imposes its own constraints: the application must expose safe reconfiguration points and support checkpoint/restart logic capable of restoring the execution state after a resize. The resulting trade-off is well captured by the experimental evidence of Section 5.6: malleability usually behaves as a compromise between baseline and workflows, preserving much of the wall-time behavior of the former while recovering a substantial fraction of the resource savings of the latter. Under contention, this can translate into a particularly favorable balance between throughput and time-to-solution.

The importance of quantum-job granularity provides a second key axis for comparison. When quantum phases are very short and frequent, as in workloads resembling gate-based offloading to fast superconducting devices, infrastructure-level multiplexing becomes especially relevant, because the dominant inefficiency lies in the inability of the scheduler to exploit idle intervals between bursts of quantum activity. In contrast, when quantum phases are longer and create substantial waiting windows, dynamic strategies that explicitly release classical resources become increasingly advantageous. This is precisely the regime in which workflow decom-

position and malleability show the clearest gains over a static baseline. Figure 5.12 is helpful here as well, since it visually associates vQPUs with the finer-grained end of the spectrum and workflows/malleability with progressively coarser, more explicit control over hybrid phases.

A further distinction concerns programmability and separation of concerns. vQPUs are the least intrusive solution from the user perspective, because the application can remain unchanged and the optimization is absorbed by the infrastructure layer. Workflows require the most explicit reformulation of the program, but they also offer the clearest separation between application logic and execution policy, which is valuable in heterogeneous and multi-backend settings. Malleability requires less structural decomposition than workflows, yet it shifts complexity into runtime-aware application design, where correctness depends on the disciplined management of state across reconfigurations. In this sense, the three strategies trace a continuum: from infrastructure transparency, to orchestration-level explicitness, to runtime-level adaptivity. This continuum aligns with the architectural requirements set out earlier in the Chapter, especially the need to support multiple integration modes while preserving separation of concerns among application, runtime, and scheduler.

An important conclusion of this comparative analysis is that these strategies are not mutually exclusive. On the contrary, they can be interpreted as composable mechanisms operating at different layers. A workflow-managed application may still benefit from vQPU-based sharing of the physical quantum device. Likewise, malleable classical phases could coexist with scheduler-level quantum virtualization, so that both classical and quantum inefficiencies are reduced simultaneously. This layered view is arguably the most coherent interpretation of the Chapter as a whole: the problem of hybrid QC–HPC execution should not be reduced to a single scheduling trick, but should instead be approached through coordinated mechanisms that reflect the actual structure of the workload and the actual scarcity profile of the platform resources.

Table 5.3 provides a compact reference for these distinctions. The table makes explicit that no single strategy dominates all dimensions: the preferred solution depends on which row matters most for the target deployment.

Overall, the evidence collected in this Chapter suggests the following synthesis. If the goal is to improve utilization of a scarce QPU while preserving complete transparency to applications, vQPUs provide the most natural solution. If the goal is

Table 5.3 Comparative summary of the three hybrid QC–HPC execution strategies. Rows capture the dimensions most relevant to deployment decisions; no single strategy dominates across all dimensions.

	vQPUs	**Malleability (DMR)**	**Workflow (StreamFlow)**
Software stack level	Infrastructure	Scheduler / runtime	Orchestration
Overhead on individual job	Quantum queue time (small, tunable)	Reconfiguration cost (small for shrinking)	Slurm queue (variable)
Resource efficiency			
– single job	Neutral	Moderate savings	High savings
– cluster-wide (QPU)	High	Moderate	Moderate
Best workload profile	Short QC, medium-long HPC	Medium-long QC, medium-long HPC	Medium-long QC, medium-long HPC
Applicability	Any hybrid job (transparent)	MPI applications (few LOC changes)	WMS-ready applications (may require decomposition)

to minimize classical resource waste and the application can be modularized cleanly, workflow decomposition is the most resource-efficient option. If the goal is to retain a single-job programming model while still adapting allocations to phase changes, malleability offers the most balanced compromise. The comparative value of each strategy therefore emerges not in isolation, but from the match between workload structure, platform constraints, and optimization objective. This observation also motivates the transition to the next Chapter, where these execution strategies are evaluated in concrete experimental settings and related back to the broader questions of performance, scalability, and resource utilization in hybrid QC–HPC systems.

Chapter 6

Experimental Evaluation

This Chapter reports the experimental evaluation of the hybrid quantum–classical methodologies introduced in Chapters 3 and 4. While the previous Chapters focused on algorithmic structure, modeling choices, and hardware-aware design principles, the objective here is to assess how those methods behave when instantiated on concrete datasets, execution environments, and quantum backends. The chapter therefore complements the methodological discussion with an empirical analysis spanning Quantum Machine Learning and combinatorial optimization, with particular attention to the interplay between algorithmic performance, hardware noise, and execution constraints.

More specifically, the Chapter considers the three case studies introduced in Section 3.3.4, 3.4.4 and 4.5. The first and second ones are QML applications on analog neutral-atom devices, namely credit-card default prediction through Quantum Reservoir Computing and graph classification through the Quantum Evolution Kernel. The latter belongs to the optimization domain and concerns the BBQ-mIS methodology for graph coloring on neutral-atom hardware. Together, these case studies cover markedly different hybrid execution patterns: iterative feature extraction followed by lightweight classical learning in the QRC setting, kernel acquisition and subsequent classical classification in the graph-learning setting, and state-dependent combinatorial search with repeated quantum sampling in the graph-coloring setting. This variety is useful because it makes it possible to evaluate not only the quality of the obtained results, but also the broader implications of hardware characteristics and access models on different families of NISQ workloads.

A central aspect of the experimental campaign is that all quantum resources were accessed under remote-service conditions rather than through tightly integrated local accelerators. At the time when these studies were carried out, there were no production grade hybrid clusters. Indeed, the target quantum platforms were available only through managed remote interfaces, and cloud-like submission was effectively the only viable operational mode for executing experiments on real hardware. For this reason, the results discussed in this Chapter should be read not as idealized device-level benchmarks, but as measurements obtained under the realistic execution conditions that characterized early NISQ experimentation, including remote job submission, backend queueing, limited shot budgets, and asynchronous result retrieval.

6.1 Experimental Platforms and Testbeds

The experimental campaign relies on a heterogeneous set of platforms, reflecting the different roles played by classical and quantum resources during development, calibration, and final execution. Across all case studies, classical resources were used both for data preprocessing and post-processing and, when necessary, for noiseless or shot-based emulation of the target quantum dynamics. Real-hardware executions were instead performed on QuEra's Aquila neutral-atom platform, accessed remotely through managed services. The resulting setup is intentionally heterogeneous: it mirrors the practical conditions under which hybrid quantum–classical applications were actually deployable during the NISQ period.

6.1.1 General execution model

From a high-level perspective, the experimental workflow adopted in this Chapter follows a common pattern. First, each methodology was developed and validated in a classical environment, where the absence of hardware noise and the greater controllability of the execution allowed one to identify meaningful hyperparameters and establish reference behaviors. Second, selected subsets of the corresponding workloads were executed on real hardware, with the purpose of assessing feasibility and quantifying the effect of realistic device-level imperfections. This two-stage process was not merely convenient, but necessary: the limited availability and cost of

quantum resources prevented exhaustive end-to-end experimentation directly on the QPU. Therefore, extensive design-space exploration had to be performed offline on classical systems, reserving hardware runs for the most informative configurations and for the final comparative assessment. In this sense, the classical platforms used in this Chapter should not be interpreted only as baselines, but also as indispensable components of the overall experimental stack.

6.1.2 Platforms for the QRC case study on credit-card default prediction

The credit-default case study combines classical preprocessing and classification with a QRC feature-extraction layer implemented on a neutral-atom simulator. As mentioned in Section 3.3.4, the reference dataset [93] is originally composed of 30,000 samples and 23 explanatory variables. After data cleaning, the effective dataset contains 29,601 samples, and two nested subsets are further extracted to fit the hardware budget, namely `CARDS_2500` with 2,571 samples and `CARDS_1000` with 1,000 samples. These subsets preserve class proportions and allow the hardware campaign to be carried out under constrained shot availability while still remaining statistically meaningful. The results are reported in Section 6.3.1.

The quantum platform for this study is QuEra's Aquila 256-qubit neutral-atom simulator. Two encoding strategies are considered, namely position encoding and local-detuning encoding, both leveraging the analog Hamiltonian of the device. In practical terms, Aquila is used as the backend for the reservoir layer only, while the rest of the pipeline, including preprocessing, resampling, and final classification, remains classical. Since exhaustive hardware evaluation on the full dataset would have been prohibitively expensive, the real-device runs are restricted to the smaller `CARDS_1000` and `CARDS_2500` subsets, whereas the larger `CARDS_30000` dataset is used mainly for noiseless emulation and for establishing a reference performance profile.

On the classical side, the emulation of the neutral-atom dynamics is performed with the Julia implementation of the Bloqade framework, which provides both noiseless statevector-level evolution and shot-based simulation of measurements. This emulation environment is used to tune the relevant parameters of the two encoding schemes and to quantify the gap between idealized and noisy execution.

The resulting setup is therefore tripartite: noiseless emulation, shot-based classical emulation, and real hardware on Aquila.

6.1.3 Platforms for the QEK case study on graph classification

The graph-classification case study targets the `PROTEINS` benchmark dataset, where each graph represents a protein and the binary label distinguishes enzymes from non-enzymes. Since the direct execution of the full dataset on a neutral-atom backend is constrained by both the available number of qubits and the geometric embedding restrictions of the hardware, the original dataset is filtered according to two criteria: the graph must fit within Aquila's 256-qubit register, and it must admit a feasible Unit-Disk embedding under the hardware constraints. This filtering yields two subsets, denoted `PROTEINS12` and `PROTEINS256`, with 143 and 276 graphs, respectively. The results of the preprocessing phase is detailed later in Section 6.2.3.

Also in this case the real backend is Aquila, accessed remotely. The smaller subset, `PROTEINS12`, is used both for classical emulation and for the optimization of the pulse parameters defining the layered time evolution of the Quantum Evolution Kernel. The larger subset, `PROTEINS256`, is then employed for the hardware evaluation. From the classical point of view, the pulse-optimization stage and the early emulation experiments are carried out on a standard laptop environment. This emphasizes an important practical aspect: even when the final workload targets a many-qubit analog device, a substantial part of the experimental design loop can still be executed on relatively modest classical resources, provided that the selected benchmark subset is sufficiently small.

As mentioned in Section 3.4.4, the graph-to-register mapping is achieved through a dedicated embedding procedure based on the constrained Unit-Disk graph formulation described in Appendix A.1. This embedding stage is part of the classical preprocessing and is required to make the graph connectivity compatible with the distance-dependent interactions available on the neutral-atom device. The resulting testbed therefore includes a nontrivial classical preparation layer, a classical emulation layer for pulse selection, and a remote hardware layer for the final execution of the kernel acquisition phase.

6.1.4 Platforms for the BBQ-mIS case study on graph coloring

The optimization case study follows a somewhat different logic. The general BBQ-mIS methodology was originally validated through large-scale classical emulation and HPC-supported exploration of graph instances, but the portion that is relevant for the present chapter concerns the execution of representative MIS subproblems on real neutral-atom hardware. More precisely, the real-hardware campaign does not consist in running the entire B&B loop online on the QPU. Instead, it focuses on a set of subgraphs extracted from the solution path of a representative twenty-node graph-coloring instance derived from antenna positions in the city of Turin. This is a direct consequence of the lack of tight classical–quantum integration. In fact, the B&B approach and implementations, such as the `pybnb`[1] python library, cannot be easily decomposed into a workflow-like execution, as they are built and optimized for the HPC environment. The asynchronous type of interaction typical of cloud-like access requires a long waiting time (often on the order of hours/days) for each submitted quantum jobs, which is incompatible with the B&B iterative execution model.

The classical side of this experimental stack is therefore split into two distinct roles. First, it includes the environments used for the emulation and for the identification of informative subproblems along the BBQ-mIS exploration tree. Second, it includes the local classical infrastructure needed to prepare the analog programs sent to the QPU. In the source material, the Bloqade framework is used to emulate the neutral-atom dynamics and to generate pulse programs, while the overall graph-coloring methodology is developed in a broader classical environment that also exploits standard graph-processing tools and exact classical solvers for reference analysis.

The actual hardware execution is again performed on Aquila, accessed through Amazon Braket. Three tasks are launched, corresponding to three subgraphs denoted `graph_1`, `graph_2`, and `graph_3`, extracted from successive nodes of the BBQ-mIS solution path. The lattice configurations and pulse waveforms used in these tasks are chosen to match those used in the corresponding emulations as closely as possible, allowing for a direct comparison between the solution histograms generated in the noiseless setting and those produced by the real device. Each task is configured nom-

[1] https://pybnb.readthedocs.io/en/stable/

inally for 1,000 shots, although only the successfully initialized shots are retained in the final analysis.

6.1.5 A note on remote access and practical reproducibility

Across all three case studies, the quantum hardware was accessed through remote-service interfaces rather than tightly coupled local accelerators. This is not incidental: the platform contributes to the observed behavior through submission constraints, shot limits, and backend-side operational policies, and should be understood as an integral part of the experimental setup rather than an inconvenience to be factored out.

6.2 Detailed Methodology

Given the heterogeneity of the considered use cases, the evaluation methodology is organized around a common comparative principle rather than a single application-specific protocol. In all cases, the goal is to compare three levels of execution realism: an idealized classical emulation or benchmark condition, an intermediate shot-based or noisy proxy when available, and the final execution on remote neutral-atom hardware. A terminological note is in order: throughout this chapter, *emulation* refers to the classical computation of the quantum dynamics (via statevector or shot-based methods on conventional hardware), while *simulation* denotes execution on the real analog quantum simulator (Aquila). This layered comparison makes it possible to separate the contribution of the algorithmic design from the effects induced by sampling noise, decoherence, limited shot counts, and hardware-level imperfections.

At the same time, the selected metrics are adapted to the structure of each problem. For the QML case studies, the emphasis is on predictive quality and on the stability of feature extraction under noise. For the optimization case study, the focus shifts to the consistency of the solution distributions produced by the quantum device with respect to the corresponding emulations and to the usefulness of the sampled maximal Independent Sets for the higher-level graph-coloring workflow. The present section therefore introduces first the common methodological logic and then the specific metrics adopted for the individual application families.

6.2.1 Common evaluation methodology

The general methodology can be summarized as follows. For each case study, the quantum-enhanced routine is first analyzed on classical resources, either in the form of noiseless emulation or through a controlled simulation procedure that isolates sampling effects. This stage serves two purposes: it provides a baseline against which hardware results can be compared, and it allows one to select suitable operating parameters without consuming excessive quantum resources. The real-hardware campaign is then performed only on those configurations that have been deemed representative or promising at the classical stage.

This methodology is particularly evident in the QEK study, where the pulse parameters are optimized on `PROTEINS12` before being transferred to Aquila, and in the QRC study, where the larger credit-default dataset is used mainly for emulation and classifier screening, while the hardware executions are restricted to smaller subsets. In the BBQ-mIS study, the same principle appears in a slightly different form: the full combinatorial workflow is not rerun end-to-end on the QPU, but specific MIS subproblems, identified through the classical exploration, are executed on Aquila to assess whether the hardware returns distributions that remain useful for the classical Branch-and-Bound logic.

A second common aspect concerns the use of comparative baselines. In all three studies, the quantum-enhanced method is not evaluated in isolation. Instead, it is systematically compared with one or more classical references. In the QRC study, the comparison includes fully classical pipelines, classical reservoir computing, and a deep neural network. In the graph-classification study, the QEK is compared with the Shortest Path Kernel [103]. In the graph-coloring study, the real-hardware histograms are compared against Bloqade-based emulation results and interpreted in the context of the broader BBQ-mIS workflow. This comparative design is important because it prevents the discussion from collapsing into a device-only benchmark and instead situates the quantum component within the larger hybrid application.

6.2.2 Methodology for the QRC credit-default study

For the credit-default case study, the evaluation pipeline is organized as a sequence of preprocessing, reservoir-based feature extraction, and downstream binary classifi-

cation. Since the general rationale of Quantum Reservoir Computing has already been introduced in Chapter 3, the purpose of this subsection is to clarify the specific protocol used for the experiments and the way in which the different baselines were constructed. In particular, the study was designed so as to compare three closely related settings: a Fully Classical (FC) pipeline operating directly on preprocessed features, a reservoir-computing pipeline in which the nonlinear feature map is provided either by a quantum or by a classical reservoir, and a deep neural network used as a nonlinear classical benchmark. This design makes it possible to separate the contribution of the reservoir representation from that of the final classifier and to quantify the effect of hardware noise on the overall hybrid workflow. A schematic view of the full pipeline is reported Figure 3.1.

The benchmark task is credit-card default prediction on the Taiwanese credit-card dataset, a widely used imbalanced binary-classification problem. The original dataset contains 30,000 samples, each described by 23 variables and a binary label indicating whether the cardholder defaults in the following month. The features include credit information, demographic variables, payment-status history, bill statements, and previous payments. After cleaning, the effective dataset contains 29,601 samples and is denoted as `CARDS_30000`. Since executing the full dataset on the neutral-atom backend would have required an impractical amount of quantum resources, two smaller nested subsets were also considered, namely `CARDS_2500` with 2,571 samples and `CARDS_1000` with 1,000 samples, both obtained through stratified extraction in order to preserve the class distribution. The resulting relation is

$$\texttt{CARDS_1000} \subseteq \texttt{CARDS_2500} \subseteq \texttt{CARDS_30000}.$$

This choice is methodologically relevant because it allows one to test the hypothesis that reservoir-based methods can remain useful even in limited-data regimes, while still retaining a larger reference dataset for classical emulation and model selection.

The preprocessing stage was designed to maximize the informativeness of the encoded features while keeping the dimensionality compatible with the quantum hardware budget. The first step consisted of data cleaning. By inspecting the dataset documentation and the actual entries, the study identified a number of inconsistencies in the categorical attributes `MARRIAGE` and `EDUCATION`. In particular, `MARRIAGE` contained undocumented values equal to 0, while `EDUCATION` contained undocumented values equal to 0, 5, and 6. These samples were removed, since

their number was relatively small and their exclusion did not significantly alter the dataset distribution. The payment-history variables `PAY_1` to `PAY_6` also contained unexpected values; however, because those cases appeared in much larger numbers, they were retained rather than removed. After this cleaning phase, the remaining dataset was used as the basis for the subsequent transformation steps.

The second preprocessing step consisted of feature scaling and categorical handling. Numerical features were scaled to the interval $[-1, +1]$ through an absolute-value scaling procedure. For the categorical variables, a dummy-variable encoding was adopted for `SEX` and `MARRIAGE`, which do not admit a natural ordering, while `EDUCATION` was retained as a numerical variable in order to preserve its ordinal structure and avoid an unnecessary expansion of the feature space. As a result, the initial number of features increased from 23 to 24. This transformed representation then served as input to a PCA-based dimensionality-reduction and feature-grouping procedure, with the ultimate goal of supporting feature extraction through the correlation circle method [137].

Principal Component Analysis was not used here as a replacement for the reservoir, but rather as a hardware-aware compression step. The number of retained principal components was selected iteratively until the inclusion of an additional component increased the explained variance by less than 0.001. This yielded 11 principal components, which together explain approximately 99% of the dataset variance. The corresponding explained-variance curve is reported in Figure 6.1.

The PCA scores of the original 24 features were then used as inputs to an agglomerative hierarchical clustering procedure operating in principal-component space. The clustering threshold was set to half the maximum pairwise distance among features, which led to the identification of 12 feature clusters. Features belonging to the same cluster were finally aggregated by averaging their scaled values, yielding a compact 12-dimensional feature vector for each sample. Figure 6.2 displays each feature vector projected into the 2D space defined by the first two principal components, where the features belonging to the same cluster are indicated by the same color.

This reduction step is important for two reasons: it removes redundant linear information and, at the same time, reduces the number of degrees of freedom that must be encoded on the QPU, thereby enabling more efficient sample replication within the available register.

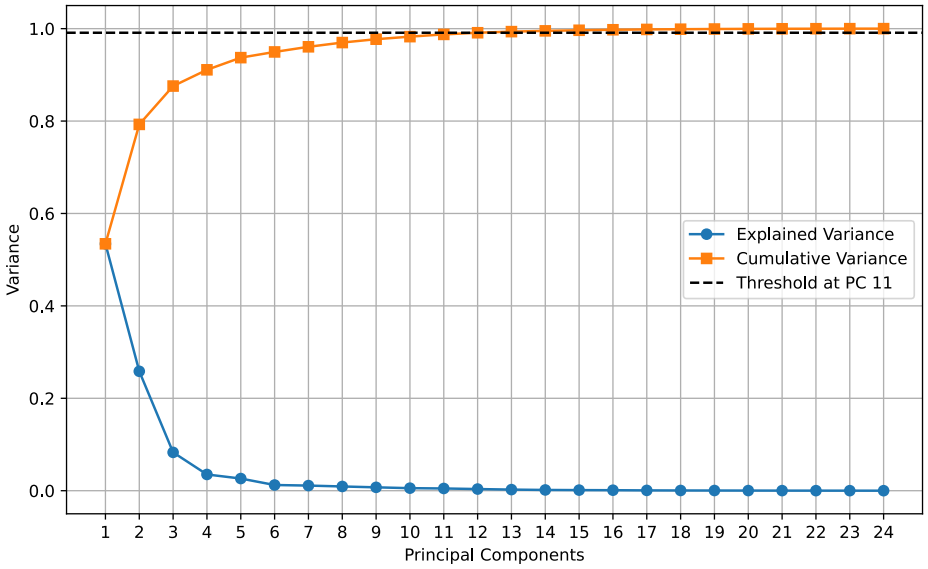


Fig. 6.1 PCA explained variance per component, computed on the `CARDS_30000` dataset. Components are ordered by descending explained variance; 11 components are retained, together explaining approximately 99% of the total variance.

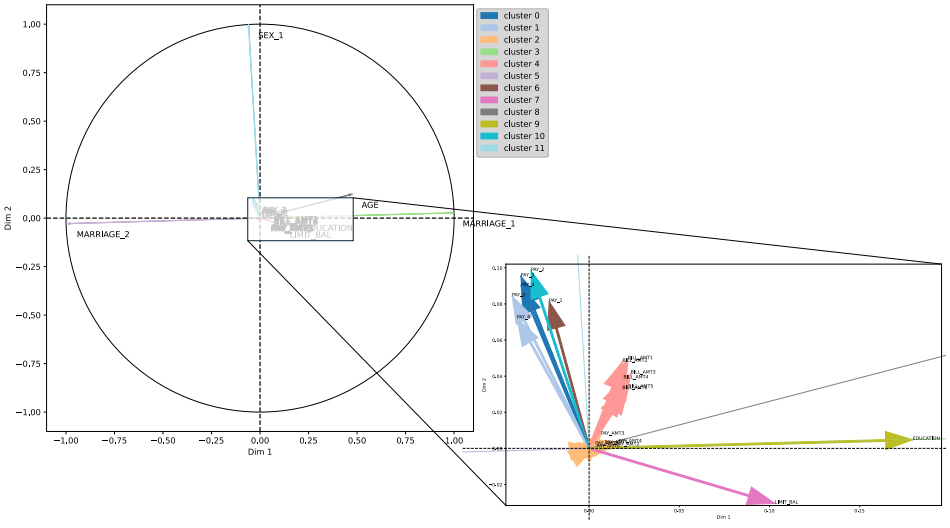


Fig. 6.2 Correlation circle for the 24 pre-processed features of `CARDS_30000`, projected onto the first two principal components. Features sharing a colour belong to the same agglomerative cluster; the 12 resulting clusters are subsequently averaged to form the 12-dimensional encoded input to the reservoir layer.

After preprocessing, the resulting 12-dimensional representation is passed to the RC layer. In the quantum setting, the reservoir is implemented on QuEra's Aquila platform by exploiting the analog dynamics of neutral atoms in the Rydberg regime. Two encoding strategies are considered. In the first one, termed *position encoding*, the information carried by the features is embedded into the distances between neighboring atoms. More precisely, if f_i denotes the i-th aggregated feature and r_0 is a reference spacing, then the distance between atoms i and $i+1$ is modulated as $r_{i,i+1} = r_0(1+\lambda f_i)$, where λ is a scaling coefficient. Since the encoded information lies in the geometry of the register, the detuning term remains global and uniform across sites. This encoding requires $n+1$ atoms for an n-dimensional sample. In the second strategy, termed *detuning encoding*, the atomic positions are kept fixed and the feature values are instead mapped to local detuning amplitudes according to $\Delta_i(t) = \Delta_{\text{global}}(t) + \Delta_l f_i$, where Δ_l is the local scaling coefficient. In this case, the number of atoms matches the number of features.

In order to construct the output feature space of the reservoir, the quantum state is not used directly. Instead, for each input sample, the reservoir is initialized in the state $|\psi_{t=0}\rangle = |0\ldots0\rangle$, evolved under the analog Hamiltonian described by Equation 3.7, and then probed at a discrete set of timesteps through a collection of local and pairwise observables. More precisely, the output vector includes the expectation values of the single-site operators Z_i and of the two-body correlation operators Z_iZ_j, with $i<j$, evaluated at each timestep. Therefore, the reservoir representation combines local excitation information with pairwise correlation information, so as to enrich the nonlinear embedding produced by the analog evolution. This choice is important because it allows the classifier to access not only marginal responses of individual atoms, but also feature-dependent interaction patterns generated by the reservoir dynamics.

Let m denote the number of atoms used to encode a sample. At a given timestep, the number of output observables is then

$$m + \binom{m}{2},$$

that is, the sum of m single-body expectation values and $\binom{m}{2}$ pairwise correlations. In the present case, the preprocessing stage yields $n=12$ aggregated input features. For the detuning encoding, the number of atoms coincides with the number of features,

namely $m = n = 12$. Consequently, each timestep produces

$$12 + \binom{12}{2} = 12 + 66 = 78$$

output features. Since the evolution is sampled at five timesteps, the overall output dimension becomes

$$78 \times 5 = 390.$$

For the position encoding, one additional atom is required, hence $m = n + 1 = 13$. In this case, the number of observables per timestep is

$$13 + \binom{13}{2} = 13 + 78 = 91,$$

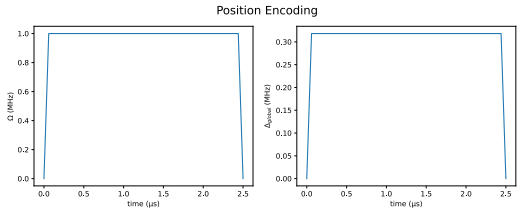
which leads to a total output-space dimension equal to

$$91 \times 5 = 455.$$

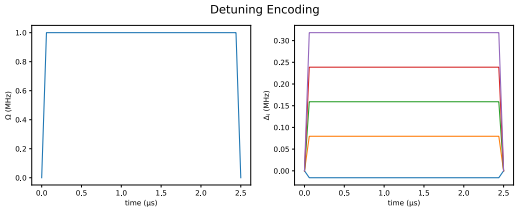
Accordingly, although both encodings start from the same 12-dimensional preprocessed input, they produce reservoir feature spaces of different sizes, with the position-encoding variant generating a slightly higher-dimensional representation because of the extra atom introduced by the geometric mapping.

The same logic is retained in the Classical Reservoir Computing baseline, where the quantum observables are replaced by the corresponding classical spin quantities. In that case, the reservoir outputs are built from the z-axis spin projections and their pairwise products at each timestep, so that the dimensionality and downstream interface of the classical and quantum reservoirs remain directly comparable. This design makes the comparison between CRC and QRC methodologically cleaner, since differences in predictive performance can be attributed to the reservoir dynamics rather than to a mismatch in the structure of the generated feature vectors.

In both schemes, the atoms corresponding to a sample are logically arranged along a one-dimensional array so that the relevant interactions are predominantly nearest-neighbor. Examples of arrays for both encodings are provided in Appendix C.1.2, together with technical details of the procedure. Figure 6.3 reports an example pulse sequence for both types of encoding.



(a) Position encoding pulse sequence.



(b) Detuning encoding pulse sequence (5-qubit array shown for readability).

Fig. 6.3 Example pulse sequence for the (a) position encoding and (b) detuning encoding. In the latter case, we considered a short 5 qubit array for the sake of readability. The relative local detuning waveforms are plotted using different colors.

The reservoir dynamics are sampled over a short analog evolution chosen to remain compatible with the typical coherence times of the platform and to reduce the impact of hardware noise. In the reported implementation, the evolution is discretized into five timesteps of $0.5\,\mu$s each, for a total simulated duration of $2.5\,\mu$s. At each timestep, observables derived from the register state are collected to form an expanded feature representation. In this way, the role of the reservoir is not to solve the classification task directly, but to map the compressed classical input into a richer nonlinear space in which simple downstream classifiers can operate more effectively. This point is central to the overall methodology: the trainable part of the model is intentionally kept lightweight, while the complexity of the representation is delegated to the fixed dynamics of the reservoir.

In order to distinguish the benefit of the quantum dynamics from that of reservoir computing more generally, the study also includes a Classical Reservoir Computing baseline. This CRC model is a dequantized analogue of the neutral-atom reservoir, obtained by replacing qubits with classical unit vectors and evolving a corresponding classical spin system under the same parameterization. In particular, starting from the Hamiltonian form used for the quantum system, a classical dynamical system is derived in terms of spin vectors $\hat{S}_i$, whose time evolution is solved numerically. The reservoir outputs are then formed from the z-axis spin projections and their products, exactly as the quantum observables are used in the QRC pipeline. Operationally, the CRC module is inserted into the same workflow as the QRC one: the preprocessed sample is fed to the reservoir, and the resulting transformed features are passed to the same family of classifiers. This baseline is important because it allows one to understand whether the observed gains come from the reservoir-computing paradigm itself or from the specifically quantum nature of the feature map.

The final stage of the methodology concerns binary classification. Because the credit-default dataset is markedly imbalanced, with a defaulter-to-non-defaulter ratio of approximately 0.22 across all considered subsets, a resampling step is introduced before model training. This resampling is applied only to the training split, whereas validation and test sets are left untouched in order to avoid data leakage and to preserve an unbiased assessment of performance on realistic data distributions. Three cases are considered: no resampling, oversampling through SMOTE and K-Means SMOTE [94], and undersampling through Cluster Centroids [95]. These techniques were selected so as to expose the classifiers to balanced or partially balanced training distributions without altering the downstream evaluation protocol. Their inclusion is particularly relevant in the present context, because the final metrics are sensitive to the trade-off between false positives and false negatives, and that trade-off can change substantially depending on how the minority class is represented during training.

The classifiers used after the reservoir layer were intentionally chosen to remain consistent with the reservoir-computing principle of simple training. More specifically, the study employs K-Nearest Neighbors, Naïve Bayes, Logistic Regression, Perceptron, Stochastic Gradient Descent, and Support Vector Machines, all implemented through standard `scikit-learn` [138] routines. For models requiring hyperparameter selection, the parameters were tuned through grid search on the validation set using the F1-score as the selection criterion. This metric was preferred to accuracy because of the class imbalance. Where appropriate, class weights were also used to compensate for the bias toward the majority class. The list of classifiers with the relative tested hyperparameter ranges are summarized in Table 6.1.

Table 6.1 Hyperparameter search ranges for the six lightweight classifiers evaluated in the QRC credit-card default study. CW: class weights enabled during training.

Model	**Hyperparameter**	**Values Tested**	**CW**
KNN	k	$\{2, 3, \ldots, 15\}$	No
SGD	Epochs	$\{10, 20, \ldots, 100\}$	Yes
Naïve Bayes	None	–	No
Log. Model	C	$\{10^{-3}, \ldots, 10^{2}\}$	Yes
Perceptron	η	$\{10^{-1}, 10^{-2}, 10^{-3}\}$	Yes
SVM	C	$\{10^{-1}, 10^{0}, 10^{1}, 10^{2}\}$	Yes

In particular, the tuning explored the number of neighbors for KNN, the number of epochs for SGD, the regularization coefficient C for logistic regression and

SVM, and the learning rate η for the perceptron, while Naïve Bayes was used without hyperparameter tuning. The inclusion of multiple lightweight classifiers is methodologically useful because it shows whether the reservoir representation is robust across different decision models rather than being tailored to a single downstream learner.

Finally, to provide a stronger nonlinear classical comparator, the study also includes a Deep Neural Network benchmark trained directly on the preprocessed features, without any reservoir layer. The DNN is implemented in PyTorch [139] and consists of six fully connected layers with hidden sizes 64, 128, 256, 128, and 64, ReLU activations, and dropout regularization, followed by a two-neuron output layer for binary classification. The model is optimized with the Adam optimizer and cross-entropy loss. This benchmark serves a complementary role with respect to the fully classical and CRC baselines. Whereas the latter are used to assess the specific contribution of the reservoir transformation, the DNN provides a reference nonlinear learner with greater expressive power but also a significantly heavier training stage. The comparison is therefore not only about raw classification quality, but also about the trade-off between representational richness and training complexity.

Overall, the methodology of the QRC credit-default study can therefore be interpreted as a controlled comparison among different ways of introducing nonlinearity into the classification pipeline. The preprocessing stage compresses and regularizes the financial data into a hardware-compatible representation; the quantum and classical reservoirs provide alternative mechanisms for nonlinear feature lifting; and the final classification stage tests whether simple trainable models can exploit those lifted features effectively under class imbalance. In the broader context of this thesis, this experimental design is particularly significant because it evaluates a quantum-enhanced workflow under the practical conditions of remote neutral-atom access, where the QPU is used specifically for feature generation while the rest of the pipeline remains classical and highly modular.

6.2.3 Methodology for the QEK graph-classification study

The graph-classification case study evaluates a Quantum Evolution Kernel workflow for binary classification of protein graphs on a neutral-atom platform. In contrast with the previous subsection, where the quantum component acts as a reservoir-

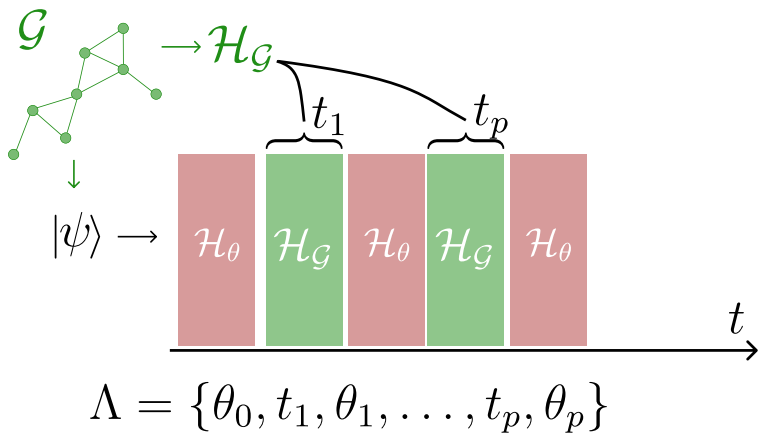


Fig. 6.4 Layered QEK time evolution: alternating application of the graph-dependent Hamiltonian $\hat{H}_G$ and the mixing Hamiltonian $\hat{H}_\theta$, parameterised by the pulse set Λ.

based feature generator for tabular data, here the quantum evolution is used to define a similarity measure directly between graphs. Since the general motivation and the high-level structure of the QEK approach have already been discussed in Section 3.4.4, the objective of the present subsection is to detail the concrete experimental protocol adopted for the benchmark, including the preprocessing and embedding stages that are required in order to execute the method on real neutral-atom hardware. Figure 6.4 summarizes the layered quantum evolution as sequence of pulses with parameterized duration on a neutral atom platform.

A first reduction of the benchmark `PROTEINS` dataset is imposed by the hardware capacity of Aquila. Since the platform provides at most 256 physical qubits, only graphs with at most 256 nodes can be considered for direct execution. A second reduction is imposed by classical emulation costs: because the simulation of the quantum evolution grows exponentially with the number of qubits, only much smaller graphs can be used during the parameter-selection stage carried out on classical resources. For this reason, the study defines two filtered and overlapping subsets. The first one, denoted `PROTEINS256`, contains all graphs with at most 256 nodes that also admit a feasible embedding on the neutral-atom register, as detailed later; it comprises 276 graphs. The second one, denoted `PROTEINS12`, is a subset of `PROTEINS256` restricted to graphs with at most 12 nodes and contains 143 graphs. The latter is used for classical emulation and parameter optimization, whereas the larger subset is employed for the final hardware evaluation. The composition of

the two embedded datasets is summarized in Table 6.2. An important detail is that PROTEINS12 is strongly imbalanced, containing 33 graphs of one class and 110 of the other, while PROTEINS256 is less skewed, with 108 and 168 graphs respectively. This imbalance motivates the use of weighted classification during training and the emphasis on the F1-score in the evaluation.

Table 6.2 Composition of the two filtered subsets of the PROTEINS benchmark used for graph classification via QEK. PROTEINS12 (graphs with $\leq$12 nodes) is used for emulation and pulse optimisation; PROTEINS256 (graphs with $\leq$256 nodes, Aquila-embeddable) is used for hardware evaluation.

	Number of graphs	#Class 1	#class 2
PROTEINS12	143	33	110
PROTEINS256	276	108	168

As mentioned, a key methodological point is that not every graph that satisfies the qubit-count constraint can be executed directly on the hardware. Neutral-atom platforms such as Aquila realize interactions through the geometric arrangement of atoms in space: two qubits effectively interact when their distance is below the blockade radius. As a consequence, the native interaction graph of the device is a Unit-Disk (UD) graph [140]. This means that, although the abstract QEK is defined for arbitrary graphs, a preprocessing step is needed in order to transform each input graph into a hardware-compatible geometric realization. As reported in [5], we considered two possible solutions. The first is a simple rescaling approach, used in [98], in which the original graph is placed in the plane with a minimum atomic spacing. However, this strongly limits the number of graphs that can fit into the physical area available on real hardware. The second approach, and the one we actually adopted, is instead to search explicitly for a UD representation of each graph. This second strategy is the only one that allows the graph connectivity to correspond directly to the Ising interaction graph of the neutral-atom device without introducing further approximations at the Hamiltonian level, and for this reason it is the one retained for the hardware-oriented evaluation in the thesis.

Neutral atom devices natively implement interactions determined by physical distance. Therefore, arbitrary graphs must be embedded as UD graphs such that an edge (i, j) exists if and only if $|\mathbf{x}_i - \mathbf{x}_j| < r_b$, where r_b denotes the Rydberg radius and $\mathbf{x}_i$ are the 2D coordinates of the i-th atom. To achieve this mapping, a dedicated neural-based embedding approach (DEN model) [9, 11] was developed and employed to solve a constrained UD graph problem. In practice, the procedure

takes an arbitrary graph and attempts to produce a feasible constrained UD graph embedding compatible with the geometry of Aquila. Relative to the original DEN formulation, additional hardware-specific constraints are introduced. First, the atoms must lie inside a rectangular register area of size $75\,\mu\text{m} \times 76\,\mu\text{m}$. Second, Aquila imposes a discrete-row arrangement, with rows separated by $4\,\mu\text{m}$. The first constraint is handled by adapting the coordinate range of the model so that generated positions lie inside the available physical area. The second is implemented through an approximation layer that snaps the coordinates to discrete rows, together with an additional penalty term in the embedding loss function that discourages pairs of atoms from being vertically closer than $4\,\mu\text{m}$. After this step, the UD constraints are re-verified to ensure that the final embedding is indeed feasible. Further details of such model are reported in Appendix A.1.

This embedding phase is not a marginal technicality, but one of the central classical preprocessing stages of the whole workflow: it determines which graphs survive the hardware-compatibility filter and, therefore, which portion of the benchmark dataset can actually be used for the quantum experiment. A representative example of an initial graph and of the corresponding feasible embedding is shown in Figure 6.5.

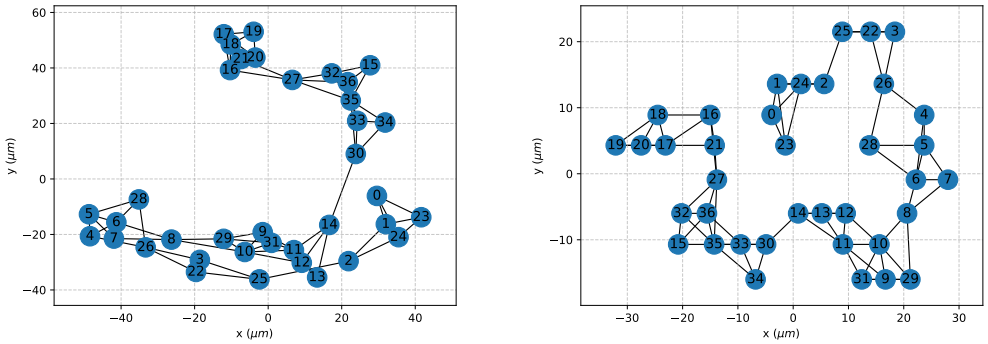


Fig. 6.5 Example of original graph (left) and corresponding feasible UD embedding under hardware constraints (right).

Once a feasible embedding has been obtained, the graph is mapped to a quantum register by associating one node to one atom. The QEK is then built through a layered time evolution governed by the two Hamiltonian contributions $\hat{H}_G$ and $\hat{H}_\theta$ described in Equations 3.10 and 3.11.

Referencing the definitions of Section 3.4.4, the overall dynamics are generated by alternating these two ingredients through a sequence of constant pulses Λ_{BO}.

Since neutral-atom devices naturally support global Rabi driving, both Hamiltonian components can be implemented natively through global pulses, provided that the graph has already been embedded in a compatible UD form. The layered time-evolution structure is illustrated in Figure 6.4.

For each graph sample G_i, the quantum register is initialized according to the corresponding embedding and evolved under the Hamiltonian determined by the selected parameters. The output of the quantum stage is not a feature vector in the standard sense, but rather a set of repeated computational-basis measurements,

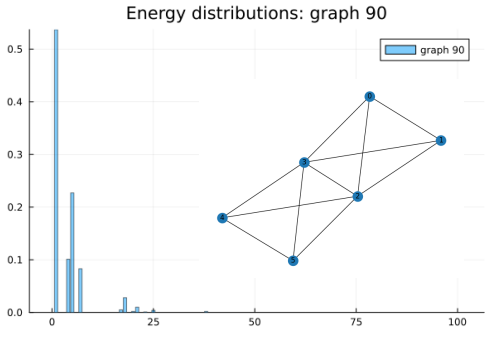
$$M_{G_i} = \{m_1, \ldots, m_k\},$$

where each measurement m_ℓ is a bitstring of length equal to the number of nodes of the graph. The bitstring components are written as n_i, with $n_i = 1$ if atom i is measured in the ground state and $n_i = 0$ if it is measured in the Rydberg state. The number of valid measurements is in general $k \leq n_{\text{shots}}$, because some shots may fail during register initialization and are therefore discarded. This point is experimentally important because it shows that the effective number of usable samples is partly determined by hardware-side behavior rather than solely by the nominal shot budget requested by the user.

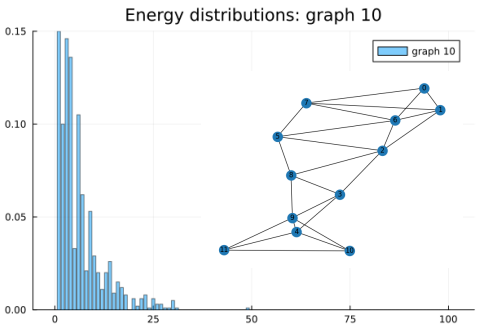
The classical post-processing stage transforms these measurement outcomes into graph-dependent probability distributions. For each shot m_ℓ, an interaction energy is computed as

$$e_\ell = \sum_{j<k} V_{jk} n_j n_k,$$

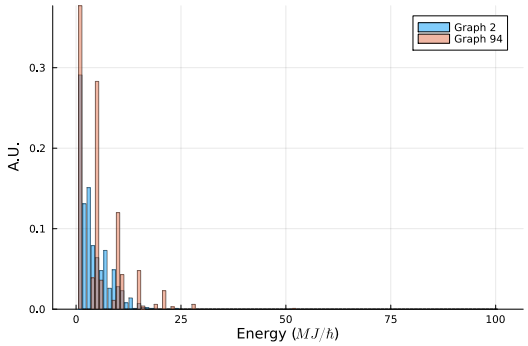
where V_{jk} denotes the van der Waals interaction coefficient associated with the pair of atoms (j,k). Repeating this procedure over all valid shots produces, for each graph, a set of measured energies. In order to make the resulting empirical distributions comparable across graphs, the energies are binned over a common support. More precisely, after collecting all graph-specific energy sets, a global minimum e_1 and maximum e_2 are extracted, and the interval $[e_1, e_2]$ is partitioned into 100 equal-width bins. Normalizing the resulting histogram by the number of valid shots yields a probability distribution P_G associated with the graph. This distribution is the fundamental object used by the QEK: rather than comparing raw bitstrings or local observables, the method compares the distributions induced by the graph-dependent



(a) Energy distribution and the associated graph.



(b) Energy distribution and the associated graph.



(c) Comparison of the energy distributions of two graphs belonging to different classes.

Fig. 6.6 Example QEK energy distributions for two graphs from the `PROTEINS12` dataset. Sub-figures (a) and (b) show the interaction-energy histogram alongside the corresponding graph structure; sub-figure (c) overlays the distributions of two graphs from different classes, illustrating the discriminative structure exploited by the Jensen–Shannon kernel.

time evolution. Examples of such energy distributions for representative graphs are reported in Figure 6.6.

Given two graphs G and G', their similarity is then computed through the kernel based on the Jensen–Shannon divergence (see Equation 3.13). The only explicit hyperparameter in this expression is μ, which controls the dynamic range of the kernel. Following the reference implementation adopted in [98], μ is fixed to 1, which keeps the kernel values in a restricted interval and avoids introducing an additional optimization loop at this stage. Once all pairwise similarities are computed, they are assembled into the kernel matrix

$$K_{i,j} = K(G_i, G_j),$$

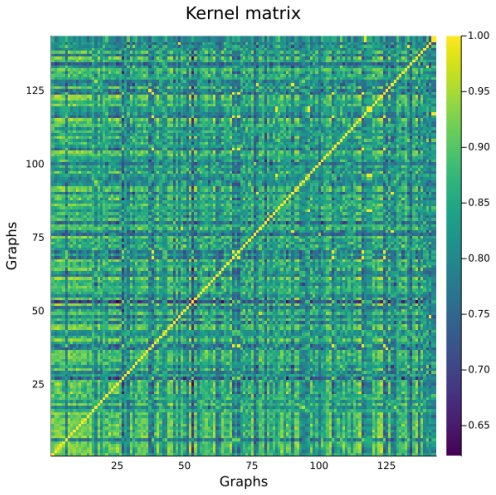


Fig. 6.7 QEK kernel matrix for PROTEINS12 (emulation, Λ_{BO}, $\mu = 1$). Each cell (i, j) encodes the Jensen–Shannon-based similarity between graphs G_i and G_j; block structure indicates class-related groupings.

which is then used as input to a classical Support Vector Machine. The kernel matrix can also be inspected visually as a heatmap. The matrix for PROTEINS12 is shown in Figure 6.7. It already shows a degree of block structure, indicating that the quantum kernel captures class-related graph similarities.

The downstream classifier is an SVM trained directly on the precomputed quantum kernel. The training protocol combines a 10-fold cross-validation scheme with a grid search over the SVM hyperparameters. In particular, two quantities are tuned. The first is the regularization parameter C, explored over 100 values logarithmically spaced in the interval $[10^{-4}, 10^{4}]$. The second is the class-weight vector w, explored through 30 pairs of the form $[1, x_i]$ with $x_i \in [1, 1000]$. The role of C is to balance margin maximization against training errors, while the class weights compensate for the imbalance present especially in PROTEINS12. For every candidate hyperparameter configuration, the model is trained on 9 folds and validated on the remaining one, and the score is averaged over the 10 splits. The F1-score is used as the optimization target because it is more informative than raw accuracy in the presence of class imbalance. This part of the protocol is worth emphasizing because it ensures that the classifier selection is not biased toward majority-class predictions and because it keeps the comparison with the classical baseline method on a fair footing.

A further methodological layer concerns the optimization of the pulse parameters Λ. Since the quality of the kernel depends strongly on the chosen time-evolution

schedule, these parameters are not fixed arbitrarily. However, optimizing them directly on hardware would be impractical, because evaluating a single candidate set requires computing the graph distributions, constructing the kernel matrix, and running the full cross-validated SVM training loop. For this reason, as mentioned in Section 3.4.4 we adopted a Bayesian optimization procedure carried out on the classically emulable subset `PROTEINS12`. In the pulse parameterization used for the optimization, the amplitude of the Rabi drive is fixed to the maximum value supported by the device, $\Omega_0 = 15.8\,\mathrm{rad}/\mu\mathrm{s}$, in order to reduce the overall evolution time and thus limit the effect of noise. The objective function maximized by Bayesian optimization is the average F1-score obtained over the 10 folds after SVM hyperparameter selection. This design reflects a genuinely hybrid optimization loop: classical emulation is used to search for waveform parameters that maximize the utility of the kernel before transferring the selected schedule to the remote quantum backend.

Overall, the methodology for this case study is therefore composed of four tightly coupled stages: hardware-aware graph preprocessing and embedding, graph-dependent quantum evolution, classical estimation of energy distributions and kernel values, and weighted SVM classification with cross-validated model selection. Reporting the preprocessing details in this subsection is useful because it makes the experimental protocol self-contained: the reader can see directly which fraction of the original benchmark survives the hardware constraints and how the resulting execution pipeline is shaped by the physical characteristics of the target neutral-atom device. This is especially important in the context of this thesis, because the hardware was accessed only through a cloud-like remote interface: under these conditions, embedding feasibility, shot efficiency, and pulse selection are not peripheral implementation details, but central determinants of which experiments can actually be run and of how informative their results can be.

6.2.4 Methodology for the BBQ-mIS graph-coloring study

The evaluation methodology for the graph-coloring case study is centered on the BBQ-mIS workflow, but it must be interpreted in a more selective way than the QML studies discussed previously. In the present Chapter, the objective is not to repeat the full algorithmic development of BBQ-mIS, which has already been introduced in Section 4.5, nor to revisit the digital QAOA-based execution pattern

employed in Section 5.4 for system-level characterization. Rather, the focus here is on the specific protocol used to assess whether real neutral-atom hardware can provide useful maximal Independent Set candidates for the classical Branch-and-Bound logic when the MIS subproblems are instantiated on an analog device and accessed under realistic remote-service conditions. This distinction is important because in this Chapter the evaluation concerns the methodological viability of the hardware-assisted component within the global hybrid workflow, not the end-to-end benchmarking of a tightly integrated online Branch-and-Bound execution.

As we discussed in the BBQ-mIS paper [6], the starting point is the observation that a direct graph-coloring encoding typically scales poorly on NISQ hardware, whereas an iterative decomposition into Independent-Set subproblems is more compatible with available resources. The key structural fact used by the methodology is that every graph admits an optimal coloring in which at least one color class is a maximal Independent Set. This makes it possible to restrict the branching space to mISs rather than exploring arbitrary Independent Sets, thereby reducing the combinatorial burden while preserving optimality at the level of the classical search logic [141, 142]. This property, already used in exact graph-coloring literature [143] and adopted explicitly in the BBQ-mIS formulation, underlies the whole experimental protocol described here, because it justifies the use of sampled mISs as branching objects rather than merely as heuristic suggestions.

At the algorithmic level, the graph-coloring problem is therefore reformulated as a sequence of MIS extractions embedded within a Branch-and-Bound procedure. Recalling the notation introduced in Section 4.5, given a graph $G = (V,E)$, each node of the search tree is associated with an induced subgraph $H = (U,I)$, a partial coloring C, the current number of used colors k, and one or more bounds on the achievable chromatic number from that node onward. In the original formulation, the state is written as (H,C,k,LB,UB), where H is obtained from the parent node by removing the vertices belonging to a selected mIS, while C contains both the color classes inherited from previous branching decisions and a worst-case completion for the still-uncolored vertices. More precisely, if the current node lies at depth l and the remaining subgraph is $H(U,I)$ with $m = |U|$, the first l color classes correspond to the maximal Independent Sets chosen along the path from the root, while the remaining m vertices are assigned one color each to define a feasible, albeit loose, completion. This representation is not repeated here for formal completeness, but because it directly determines the metrics used later in the chapter: the quality of a

sampled mIS is judged by the extent to which it improves the exploration of the state space induced by these tuples.

The practical motivation for moving from the earlier Greedy-it-MIS strategy to BBQ-mIS is also methodologically relevant. In the greedy variant, the algorithm simply selects one Independent Set returned by the MIS solver, removes its vertices from the graph, and iterates until the graph is exhausted. This guarantees a feasible coloring but does not attempt to minimize the number of colors globally, because it commits to a single choice at each step. In contrast, BBQ-mIS leverages the full measurement histogram returned by the MIS routine and interprets it as a set of branching opportunities. The histogram is therefore not treated as an auxiliary diagnostic output, but as the fundamental quantum contribution to the overall optimization process. This point is central in the methodology, because the real-hardware evaluation is explicitly designed to test whether the distribution of sampled solutions remains informative enough, under noise, to preserve this histogram-based branching logic.

Each MIS subproblem can be written in the standard binary form of Equation 4.6. In [6], we paired this formulation with a pulse-optimized QAOA-like routine on neutral-atom hardware emulated through Pulser [144], with pulse durations tuned classically and repeated measurements used to recover viable MISs from the final distribution. In this Section, we will focus on the technical details of the real-hardware experiments performed on Aquila, whose results are reported in Section 6.4. However, it is important to report the results of the aforementioned emulated procedure, since it provides the validation of the overall methodology and useful insights for the experimental campaign.

To test our algorithms, we created a dataset containing 120 samples of unit-disk graphs with a number of vertexes $n \in \{10, 11, 12, 13, 14, 15\}$. Specifically, there are 20 samples for each possible value of n. The UD feasible representation of each graph is obtained with the embedding model mentioned in Section 6.2.3.

All the experiments were run on an IBM Power9-based cluster, with 32 cores/node and 256 GB/node. In particular, each trial uses 32 physical cores of a node and used up to 16 MPI processes for the parallelization of the Branch-and-Bound. The overall experiment required ≈ 8000 core hours.

On this dataset, the BBQ-mIS algorithm always reaches the coloring solutions with the same number of colors as the optimal graph coloring solutions computed

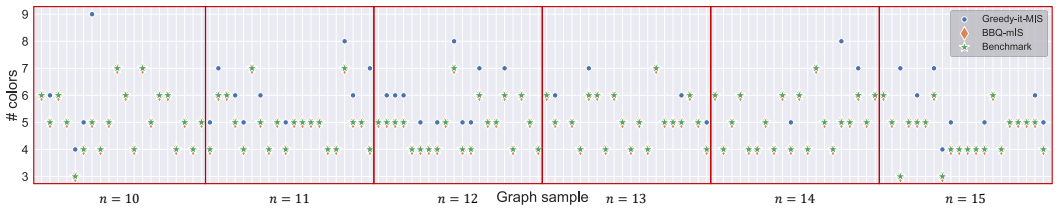


Fig. 6.8 Graph coloring results (number of colors used) for Greedy-it-MIS and BBQ-mIS on the 120-instance UD graph benchmark ($n \in \{10, \ldots, 15\}$ vertices, emulation via Pulser), compared with the Gurobi exact solver. BBQ-mIS always matches the Gurobi optimum; Greedy-it-MIS produces suboptimal colorings in 38 of 120 cases.

using the exact solver Gurobi [145]. Greedy-it-MIS instead provides worse solutions for 38 samples out of 120; in the worst case, it requires up to 4 colors more (9 colors instead of 5). Figure 6.8 summarizes all graph coloring results on our dataset.

Another observation concerns the number of nodes used in the Branch-and-Bound exploration. In the case of our dataset, the maximum number of BB nodes, i.e., 50, is never reached: BBQ-mIS terminates into leaves after having explored at most 20 nodes, in the worst-case graph instances. Reasonably, the number of mIS coming from small graphs, hence the number of children generating from each branch, is significantly lower than when targeting larger graphs.

With the overall methodology validated, we can focus on the real-hardware experimental setup. Notably, the objective is not to re-evaluate the full QAOA optimization loop, but to assess the behavior of representative MIS subproblems executed on the physical device using a fixed analog program consistent with the target Rydberg dynamics. In particular, in this case the MIS solver is based on the QAA algorithm. This choice also avoids overlap with the vQPU case study of Section 5.4 and is also more coherent with the execution conditions available during the hardware campaign, where the remote and asynchronous access model made repeated online optimization within the Branch-and-Bound loop impractical.

The bounding logic of BBQ-mIS is retained unchanged in the methodology, because it is the classical mechanism through which the utility of the sampled mISs is eventually interpreted. The lower bound associated with a node is constructed by adding the current depth to a graph-theoretic lower bound on the chromatic number of the remaining subgraph. In the original BBQ-mIS formulation, the tightest value among the Hoffman [146], Elphick–Wocjan [147], and Edwards–Elphick [148] bounds is used. With respect to our work in [6], in the real-hardware study reported

in this thesis, the lower-bound set is further enriched with the Lovász number [149], which remains computationally tractable and can be tighter in some cases. Upper bounds are instead derived from simple classical coloring heuristics, namely the greedy-coloring and Welsh–Powell bounds [150], and are used to guide the search order. These details are worth retaining in the present subsection not to restate the full Branch-and-Bound theory, but because they clarify the decision criterion behind the branch exploration order and, indirectly, why recovering several valid mISs from the histogram is more useful than recovering a single best bitstring.

The tree expansion is additionally regulated by three pruning rules that are especially important in the context of noisy hardware outputs. The first is the standard pruning by non-improvement, which discards nodes whose lower bound is already worse than the best coloring found so far. The second is pruning by unfeasibility, which removes candidate children generated from bitstrings that do not correspond either to Independent Sets or to maximal Independent Sets. The third is pruning by redundancy, which avoids revisiting identical induced subgraphs reached through different branching orders by associating a fingerprint to each remaining graph. In the real-hardware setting, where noisy measurements naturally broaden the output distribution, these pruning rules become part of the measurement post-processing itself: they filter the raw bitstring samples before those samples can influence the Branch-and-Bound logic. Methodologically, this means that the evaluation is not based on raw device accuracy alone, but on the amount of *usable* combinatorial information that survives after feasibility and maximality checks.

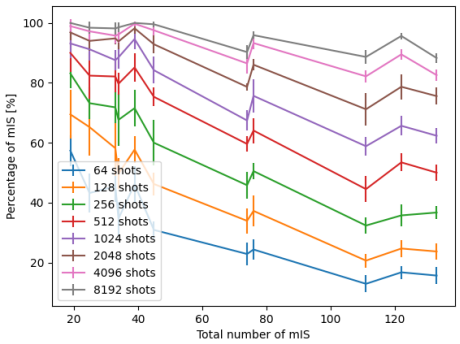
A further relevant aspect of the methodology concerns the exploration policy of the Branch-and-Bound tree. Several standard node-selection policies were considered, including FIFO, gap-based, and depth-first exploration, but we decided to define a custom priority score that balances early depth exploration with progressively more selective bound-driven branching. In [6], this score is expressed as a function of the local upper bound and the number of residual edges, whereas for the work reported in this thesis we used a closely related prioritization logic aimed at quickly obtaining a feasible incumbent while progressively steering the search toward less connected and more promising residual subgraphs. For the purposes of this Chapter, the important point is not the exact formula itself, but the fact that the hardware-sampled mISs are not consumed by the classical layer in arbitrary order. Instead, they are interpreted within a structured search policy that favors those branch expansions that are more likely to improve the incumbent coloring. This is

another reason why the experimental protocol emphasizes distributional agreement between emulation and hardware rather than only the probability of the single most frequent solution.

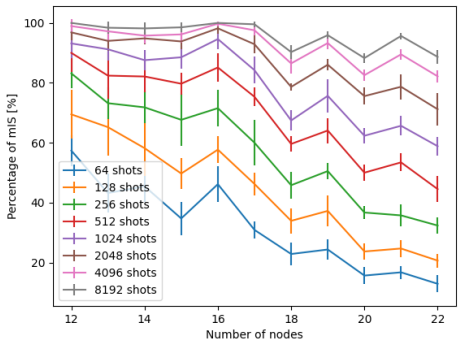
An important insight is provided by a detailed study of the number of mISs sampled by the emulated quantum algorithm (QAA) and the impact of statistical noise. To this end, we developed a dataset of graphs using the lattices provided directly by this library. Specifically, the graphs was generated using the `TriangularLattice` data type of Bloqade with a single lattice site within the UD positioned at the origin $O \equiv (0\ \mu m,\ 0\ \mu m)$ and two lattice vectors $\mathbf{v_1^T} = \begin{bmatrix} 5.0\ \mu m & 0.0\ \mu m \end{bmatrix}$ and $\mathbf{v_2^T} = \begin{bmatrix} 2.5\ \mu m & 4.330127\ \mu m \end{bmatrix}$. The connected nodes within these graphs are all those located at a distance less than or equal to 10 μm: this results in highly connected graphs with cliques reaching up to 7. Additionally, a random dropout of 33% is applied to generate more diverse graphs. In total, we generated a total of 43 graphs ranging from 5 to 22 nodes. We then executed the MIS emulated quantum solver on the generated dataset using 2^n number of shots, with $n \in \{6, 7, ..., 13\}$, 10 times for each value of n. Finally, we computed the total number of mISs for each graph using using the Julia `GenericTensorNetworks` library. Figure 6.9 show the outcome of the emulated experiments.

It is evident that the number of mISs found grows linearly with the total number of mISs. This allows us to predict that the solution tree will not explode exponentially with the increase in the number of nodes, thereby limiting the branching factor.

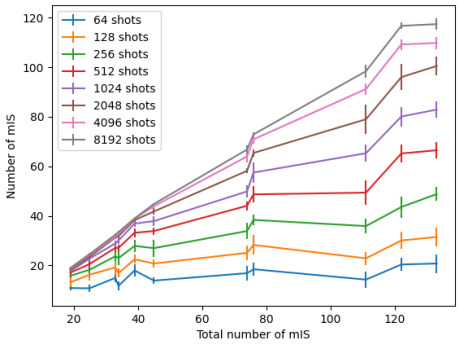
From the standpoint of this Chapter, the central metric is not the exact chromatic number obtained by the full solver, since that quantity is determined by the broader classical search and was addressed at the emulation level. Instead, the key metrics are the consistency of the sampled histograms with respect to emulation, the spread of probability mass across useful versus noisy configurations, and the number of valid mISs recovered per shot budget. Indeed, the measured metrics resulting from the experiments on Aquila are reported in Section 6.4. This shift in emphasis is deliberate. Under the remote cloud-like access conditions available at the time, the decisive question was whether the hardware could act as a reliable generator of combinatorially useful branch candidates, not whether it could be embedded transparently into a tightly synchronized exact solver.



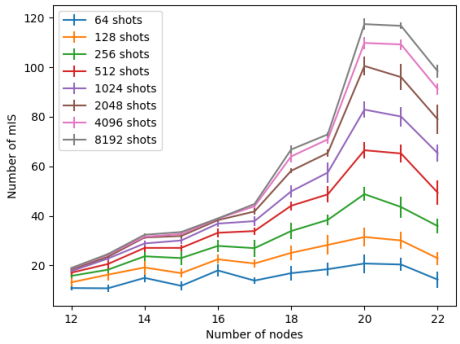
(a) Percentage of sampled mISs as a function of the total number of mISs.



(b) Percentage of sampled mISs as a function of the number of nodes of the graphs.



(c) Absolute number of sampled mISs as a function of the total number of mISs.



(d) Absolute number of sampled mISs as a function of the number of nodes of the graphs.

Fig. 6.9 Absolute number of mISs sampled by the MIS solver and the relative percentage of the total, as a function of the total number of mISs and graph size, for shot counts $n \in \{2^6, \ldots, 2^{13}\}$ (emulation only). Results are averaged over ten independent runs per graph instance.

6.3 Evaluation of QML Use-Cases

This section reports the experimental evaluation of the two Quantum Machine Learning case studies introduced in Chapter 3 and formalized in Section 6.2, namely credit-card default prediction through Quantum Reservoir Computing and graph classification through the Quantum Evolution Kernel. In line with the methodological choices discussed earlier, the objective here is not to restate the structure of the two pipelines, but rather to assess how their predictive behavior changes across classical emulation, shot-based approximation, and remote execution on Aquila. The two studies are complementary. The QRC case emphasizes feature generation followed

by lightweight classical learning, whereas the QEK case emphasizes distribution-based kernel construction followed by classical SVM training. Taken together, they provide a useful view of how different QML paradigms react to hardware noise, reduced shot budgets, and limited-data conditions.

6.3.1 Results for the classification with QRC case study

The first relevant result of the case study on QRC for default predication of credit card owners concerns the classical emulation campaign on the largest dataset, `CARDS_30000`. This phase was used both to assess whether the reservoir transformation could provide a meaningful nonlinear lifting of the data and to understand whether such a lifting remained competitive with a stronger fully classical nonlinear baseline, namely the DNN. The results visualized in Figure 6.10 show that the detuning-encoded QRC, when evaluated in noiseless emulation, reaches classification performance that is broadly comparable with the DNN and with the best fully classical alternatives, especially when combined with simple readout models such as SGD and SVM. At the same time, the comparison with the fully classical pipeline and with the CRC baseline indicates that the advantage is not due simply to the presence of a nonlinear classifier downstream, but rather to the structure of the generated feature space.

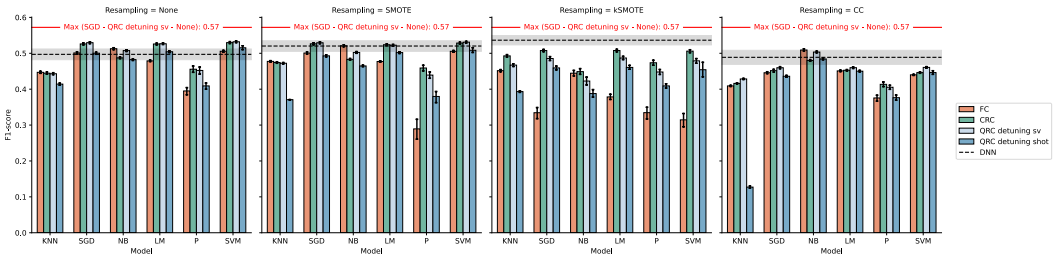


Fig. 6.10 Classification benchmark on `CARDS_30000` (noiseless emulation): emulated QRC with detuning encoding, Classical Reservoir Computing (CRC), and Fully Classical (FC) pipelines compared against the DNN. The shaded band shows the DNN standard deviation; the red line marks the maximum F1-score across all compared methods.

A second result emerging from the emulation phase concerns shot sensitivity. By comparing statevector-derived features with shot-based emulation, Figure 6.11 shows that the F1-score tends to saturate after a few hundred Hamiltonian simulations, suggesting that the statistical noise associated with moderate shot counts is not, by

itself, the dominant obstacle for the considered task. This is an important observation because it separates the effect of finite sampling from the stronger degradation induced by hardware noise in the real-device experiments. In other words, the reservoir pipeline appears reasonably robust to measurement granularity alone, at least within the shot range explored in this study.

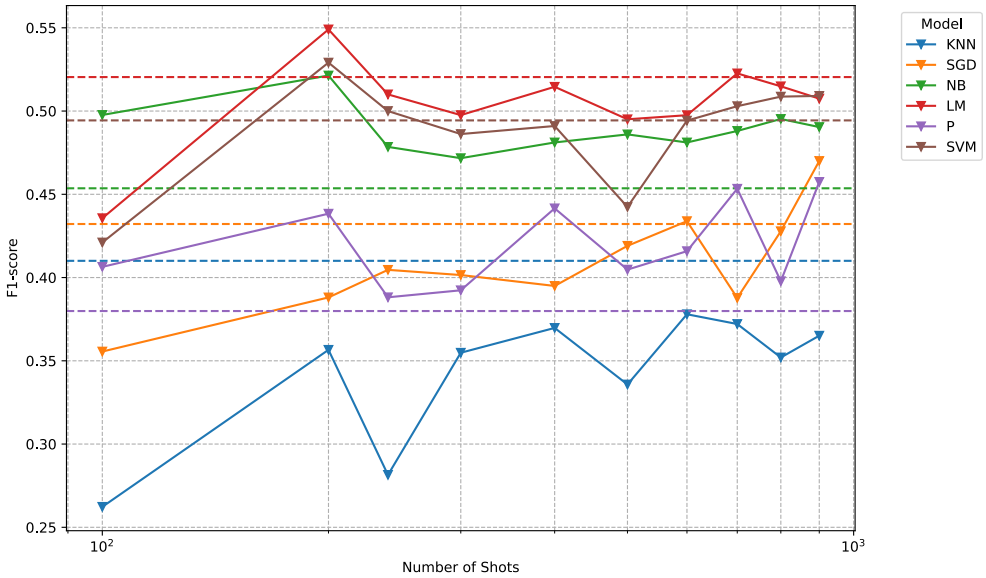


Fig. 6.11 QRC credit-default F1-score as a function of the number of Hamiltonian simulations (shot count) on `CARDS_30000` (detuning encoding). Dotted lines show the corresponding noiseless statevector emulation scores; solid lines show shot-based emulation. The score saturates after a few hundred shots, indicating limited sensitivity to the sampling budget in this range.

The transition from emulation to hardware is best illustrated by the `CARDS_1000` results shown in Figure 6.12, where both detuning and position encoding were tested on Aquila and compared against statevector and shot-based emulation. The dominant trend is a significant degradation of predictive quality when moving from noiseless emulation to real hardware. The best result on `CARDS_1000` is achieved by the detuning-encoded QRC in statevector emulation, with an F1-score of approximately 0.71 when combined with SVM and K-Means SMOTE, whereas the corresponding Aquila-based variants fall noticeably below this value. By contrast, the DNN remains comparatively stable because its entire training and inference loop remains classical. The same figure also shows that, under hardware execution, the detuning encoding is generally more robust than the position encoding. This qualitative difference is consistent with the feature-level comparison between real and emulated reservoir

outputs: the detuning scheme appears less sensitive to hardware fluctuations, plausibly because it relies on programmable local amplitudes rather than on atom-position realizations, which are more exposed to shot-to-shot geometric variability [92, 97].

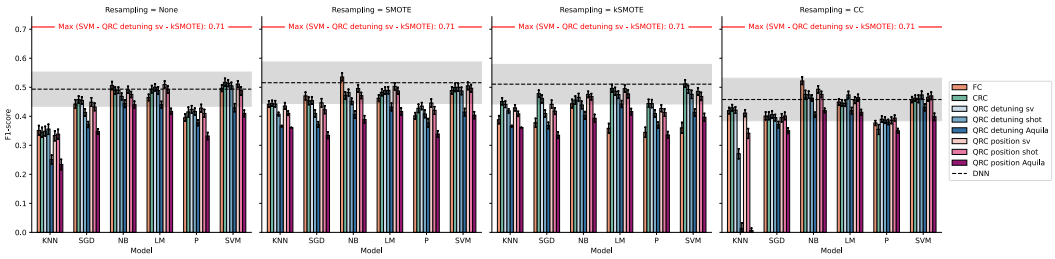


Fig. 6.12 Classification benchmark on `CARDS_1000`: real Aquila hardware and emulated QRC (both detuning and position encoding), CRC, and FC compared against the DNN. The shaded band shows the DNN standard deviation; the red line marks the maximum F1-score across all methods. Hardware results show notable degradation relative to noiseless emulation, with detuning encoding more robust than position encoding.

A particularly informative part of the QRC evaluation is the direct comparison between real and emulated reservoir features. Appendix C.1.3 reports a statistical correlation analysis between Aquila-derived features and shot-based emulation for ten randomly selected samples, as a function of both timestep and number of shots. The main conclusion is that correlation is not uniform across the reservoir evolution: the earliest timesteps retain a stronger agreement with the emulated reference, while later timesteps show a progressively larger discrepancy. This behavior is coherent with the physical picture of noise accumulation over time. Since the reservoir representation concatenates observables sampled at multiple timesteps, the final classifier receives a feature vector in which the earliest components are more faithful to the idealized dynamics and the latest components are increasingly contaminated by decoherence and hardware imperfections. The figure reporting timestep-wise correlation is therefore especially useful in the thesis because it links a device-level phenomenon directly to the observed application-level degradation.

The second round of hardware experiments on `CARDS_2500` clarifies an important point. Although hardware noise still reduces predictive quality with respect to statevector emulation, the gap becomes smaller when more data are available to the classical classifier. In the results reported in Figure 6.13, the best score on `CARDS_2500` is obtained by the CRC baseline, with an F1-score around 0.63 and no resampling, while the Aquila-based detuning QRC remains below the noiseless version but is visibly closer to it than in the `CARDS_1000` case. This suggests that part

of the noise introduced by the QPU can be statistically absorbed by the downstream classical learner when the training sample is sufficiently large. Stated differently, the QRC workflow appears to benefit from the hybrid separation of roles: even if the quantum feature generator is noisy, the classical classifier can partially learn to discount the least informative distortions when more examples are available. This is one of the most relevant outcomes of this study, because it indicates that the usefulness of the hybrid pipeline should not be judged solely from the fidelity of the quantum features in isolation, but from the behavior of the end-to-end classical–quantum stack.

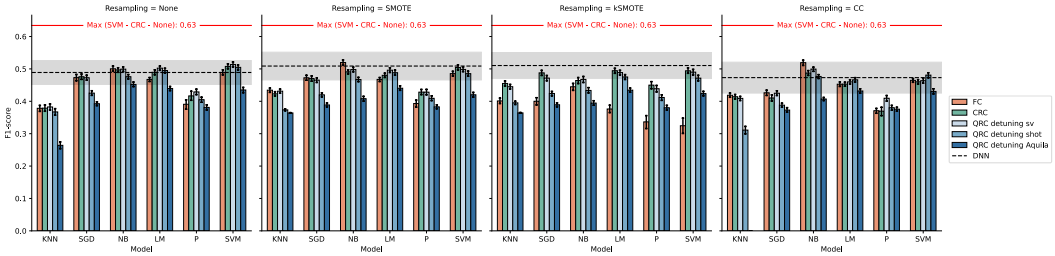


Fig. 6.13 Classification benchmark on `CARDS_2500`: real Aquila hardware and emulated QRC (detuning encoding), CRC, and FC compared against the DNN. The shaded band shows the DNN standard deviation; the red line marks the maximum F1-score. The hardware-to-emulation gap is smaller than on `CARDS_1000`, indicating that the downstream classifier partially absorbs noise with more training data.

From the standpoint of the thesis narrative, the QRC case study therefore supports three main conclusions. First, in an idealized emulation setting, the reservoir-based approach is capable of producing competitive classification performance while training only lightweight classical readout models. Second, the real-device execution on Aquila introduces a non-negligible loss of feature quality, especially on smaller datasets and for the position-based encoding. Third, this degradation is not catastrophic in the larger-data regime, where the hybrid structure allows the classical learner to compensate partially for the imperfections of the quantum feature map. This is especially relevant in the NISQ context, because it suggests that QRC may remain viable even when the quantum device is not sufficiently accurate to support end-to-end learning, provided that the quantum module is used as a fixed, noisy, but still informative feature transformer.

A further point concerns training complexity. Table 6.3 reports the training time for each combination of models and resampling methods. It can be seen that the lightweight classifiers used after the reservoir layer require substantially shorter

training times than the DNN, even when oversampling is applied. This does not imply that the full hybrid pipeline is cheaper in absolute terms once QPU time is considered, but it does highlight a structural trade-off: the QRC methodology shifts complexity away from trainable model parameters and toward the generation of a richer feature representation. In the context of this thesis, that trade-off is meaningful because it reflects one of the recurrent themes of hybrid NISQ computing, namely the replacement of deep trainable quantum or classical models with shallower trainable layers fed by more structured upstream transformations.

Table 6.4 consolidates the experimental configurations and the key quantitative outcomes across the three dataset scales and execution modes.

Table 6.3 Training times (seconds) for the classifiers evaluated in the QRC credit-card default study on `CARDS_30000`, across resampling strategies. The DNN is trained on the original 12 preprocessed features; all other classifiers use the 390-dimensional QRC output features (detuning encoding, 5 timesteps). Columns: no resampling (None), SMOTE, K-Means SMOTE (kSMOTE), Cluster Centroids undersampling (CC). Values reported as mean $\pm$ standard deviation over five runs.

Classifier	**None**	**SMOTE**	**kSMOTE**	**CC**
KNN	0.12 ± 0.01	0.23 ± 0.01	0.24 ± 0.11	0.10 ± 0.01
SGD	2.86 ± 0.34	3.43 ± 0.40	2.82 ± 0.31	1.22 ± 0.15
Naïve Bayes	0.11 ± 0.01	0.23 ± 0.01	0.20 ± 0.01	0.10 ± 0.01
Logistic Model	0.14 ± 0.01	0.31 ± 0.02	0.30 ± 0.10	0.08 ± 0.01
Perceptron	0.18 ± 0.02	0.28 ± 0.02	0.27 ± 0.01	0.09 ± 0.01
SVM	$1515.93 \pm$ 397.65	$4359.86 \pm$ 1076.96	$3976.44 \pm$ 1077.47	$150.71 \pm$ 83.96
DNN	26.52 ± 0.26	43.32 ± 2.77	42.97 ± 1.77	13.62 ± 0.62

6.3.2 Results for the QEK graph-classification case study

The QEK evaluation starts from the reduced `PROTEINS12` subset, which is used for classical emulation and pulse optimization before transferring the selected waveform to Aquila. Table 6.5 summarizes the comparison between emulation, Aquila, and SPK. In this reduced-data regime, the first relevant result is that the optimized waveform Λ_{BO} improves over the reference waveform Λ_a previously adopted in the literature [98]. On `PROTEINS12` the emulated QEK with Λ_{BO} reaches an F1-score of 46.3%, compared with 41.9% for Λ_a, while the Shortest Path Kernel reaches

Table 6.4 QRC experimental configurations and key results for credit-card default prediction. Execution modes: *SV* = noiseless statevector emulation; *Shot* = shot-based emulation; *HW* = real Aquila hardware. Qubits: 12 (detuning encoding) or 13 (position encoding); 5 timesteps; 2.5 μs total analog evolution. F1-scores reported as best mean value across classifiers and resampling strategies.

Dataset	**Samples**	**Qubits**	**Exec. modes**	**Best F1 (SV)**	**Best F1 (HW)**
`CARDS_30000`	29,601	12	SV, Shot	≈ 0.53	–
`CARDS_1000`	1,000	12/13	SV, Shot, HW	≈ 0.52	≈ 0.46
`CARDS_2500`	2,571	12	SV, Shot, HW	≈ 0.51	≈ 0.46
Classical baselines: DNN (best on `CARDS_30000`: F1 ≈ 0.53); CRC (best on `CARDS_30000`: F1 ≈ 0.53).					

44.1%. The same trend is also reflected in precision and accuracy, even though accuracy is more difficult to interpret because of the strong class imbalance of the subset. These numbers suggest that the Bayesian optimization of the pulse schedule is not merely a low-level calibration refinement, but a relevant algorithmic lever: a shorter and better-structured evolution produces energy distributions that are more useful for classification. This comparison can be documented by reusing the table reporting the `PROTEINS12` emulation results, together with the waveform comparison figure and the figure showing how the two parameter sets affect the resulting energy distributions.

Table 6.5 Classification performance on `PROTEINS12` for QEK with parameter sets Λ_a and Λ_{BO} under emulation and real Aquila hardware execution, compared with the Shortest Path Kernel (SPK) baseline.

	F-1 (%)	Accuracy (%)	Precision (%)	Recall (%)
Λ_a (Emulation)	41.9	44.8	29.3	89.1
Λ_{BO} (Emulation)	**46.3**	52.4	37.1	85
SPK	44.1	**61.4**	35	68
Λ_a (Aquila)	**48**	62.2	35.9	80.8
Λ_{BO} (Aquila)	**49**	63.7	39.5	78

An interesting aspect of the QEK study is that, on `PROTEINS12`, the noisy Aquila execution does not degrade the results with respect to emulation; on the contrary, it slightly improves them. The reported scores are 48.0% F1 for Λ_a on Aquila and 49.0% for Λ_{BO} on Aquila, both above their emulated counterparts, while the SPK remains at 44.1%. The optimized hardware-executed QEK also reaches the highest accuracy among the compared methods, namely 63.7%. A plausible interpretation

of this somewhat counterintuitive behavior is that the specific noise model of Aquila does not simply blur the distributions, but can also bias them in a way that is not entirely detrimental for the decision boundary on this small and imbalanced benchmark. Since the task is evaluated through cross-validated SVM classification on empirical kernel matrices, the relevant quantity is not fidelity to the noiseless dynamics per se, but whether the induced similarity structure remains discriminative. This observation is important because it cautions against an overly simplistic expectation that every deviation from noiseless emulation must necessarily lower downstream classification quality.

The comparison between emulated and hardware-generated energy distributions provides additional insight into this point. Figure 6.14 shows that, for representative graphs, the two distributions remain qualitatively similar, with hardware showing a somewhat larger weight on very low-energy and, simultaneously, on some higher-energy configurations. The former effect can be associated with coupling toward the ground state not captured by the idealized emulation, while the latter is linked to detection errors that can spuriously increase the apparent number of excitations. From the viewpoint of kernel construction, this means that hardware noise perturbs the empirical distributions in a structured rather than purely random manner, which can explain why the downstream kernel remain competitive despite imperfect physical execution.

The most significant application-level result is obtained on the `PROTEINS256` subset, which is the real target of the hardware campaign. As shown in Table 6.6, on this benchmark the Aquila-executed QEK with optimized parameters reaches an F1-score of 65.6%, with accuracy 65.4%, precision 55.1%, and recall 86.6%. The corresponding SPK baseline reaches 65.3% F1, 64.9% accuracy, 53.5% precision, and 87.1% recall. Thus, on the larger and less imbalanced dataset, the QEK is essentially competitive with the classical shortest-path kernel, with a slight edge in F1, accuracy, and precision, and a marginally lower recall. This result is central to the thesis because it shows that a hardware-native quantum kernel can remain relevant even at the scale of hundreds of graph nodes, provided that the graphs admit a feasible embedding on the neutral-atom device. It is also important that the optimized parameters were selected only on `PROTEINS12` and then transferred to `PROTEINS256`; despite this mismatch between optimization and deployment regime, the final hardware performance remains competitive with the classical baseline.

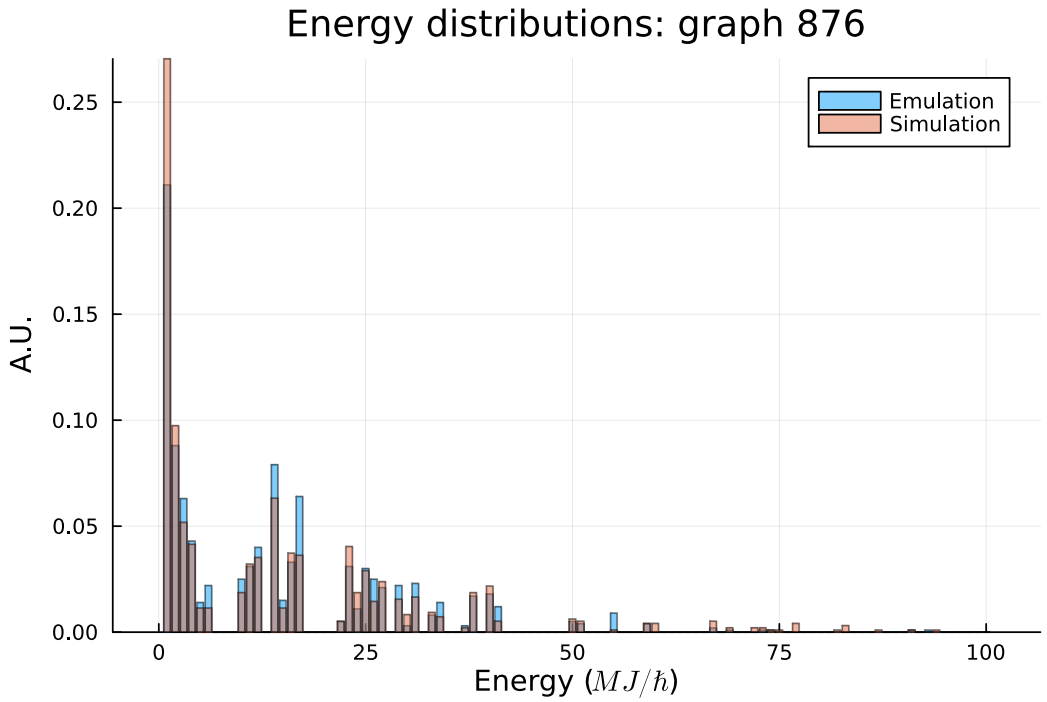


Fig. 6.14 Comparison between emulated and Aquila hardware energy distributions for a randomly selected graph from `PROTEINS12`. Hardware execution broadens the distribution relative to noiseless emulation, consistent with decoherence and readout noise.

Table 6.6 QEK (Aquila hardware, Λ_{BO}) and SPK classification performance on `PROTEINS256` (graph classification benchmark, 276 protein graphs).

	F-1 (%)	Accuracy (%)	Precision (%)	Recall (%)
Aquila simulation	**65,6**	**65,4**	**55,1**	86,6
SPK	65,3	64,9	53,5	**87,1**

A second notable result on PROTEINS256 concerns shot efficiency. Table 6.7 reports the result of a subsampling analysis in which the full set of measurements is reduced artificially to emulate smaller shot budgets. With 1000 shots, the F1-score is 65.6%; with 100 shots, it remains at 65.1%, i.e., roughly one percentage point lower on average; only when the budget is reduced to 10 shots does the performance drop more markedly, to about 59.3% F1. This indicates that the energy-distribution-based kernel is rather robust to a moderate reduction in sampling budget, which is a practically important property in cloud-like NISQ execution where shots carry both time and monetary cost. This is practically relevant because it suggests that the classification workflow does not need extremely large shot counts to remain competitive.

Table 6.7 QEK classification performance on PROTEINS256 (Aquila hardware) as a function of the number of shots used per graph for energy-distribution estimation.

number of shots	F-1 (%)	Accuracy (%)	Precision (%)	Recall (%)
10	59,3	63,1	52,9	71
100	**65,1**	64,4	53,8	87
1000	**65,6**	65,4	55,1	86,6

Overall, this case study supports a different but complementary conclusion with respect to QRC. In the QEK workflow, the quantum device is not used to produce per-sample feature vectors later denoised by a classical classifier, but to induce a pairwise similarity structure across graphs through energy distributions and Jensen–Shannon-based kernel evaluation. The experiments show that this structure remains meaningful on real hardware. On the small optimized subset, the hardware kernel slightly outperforms both its emulated counterpart and the classical SPK. On the larger benchmark, it remains essentially on par with SPK. This does not establish a broad practical advantage of the quantum method, but it does provide a convincing proof of feasibility for a hardware-native graph-learning workflow on a 256-qubit neutral-atom platform.

6.3.3 Comparative interpretation of the two QML case studies

Considered jointly, the two QML case studies reveal a useful contrast in how different hybrid learning paradigms react to NISQ imperfections. In the QRC pipeline, hardware noise primarily corrupts the generated feature vectors, and the downstream

classifier can only partially compensate for this effect. The degradation is therefore more visible when the dataset is small and the classifier has limited statistical leverage. In the QEK pipeline, by contrast, the relevant object is a distribution-derived similarity measure, which appears more stable under the type of noise introduced by the neutral-atom simulator. As a consequence, the hardware results remain closer to the classical benchmark, and in the small-data regime they can even slightly exceed the noiseless emulation.

A second contrast concerns the role of data scale. In the QRC study, increasing the number of samples from `CARDS_1000` to `CARDS_2500` reduces the relative damage induced by hardware noise, suggesting that the classical readout can learn to discount part of the distortion. In the QEK study, moving from `PROTEINS12` to `PROTEINS256` improves all performance scores substantially for both classical and quantum kernels, indicating that the main limitation of the smaller benchmark is statistical scarcity rather than an intrinsic weakness of the kernel construction itself. In both cases, larger datasets make the hybrid method more informative, but they do so through different mechanisms: improved denoising at readout time in QRC, and more reliable kernel-based discrimination in QEK.

Finally, both studies reinforce a broader point that is central to this thesis. The experimentally meaningful quantity is not the fidelity of the quantum state evolution in isolation, but the performance of the complete hybrid learning stack under realistic remote-access conditions. In one case, this means asking whether noisy reservoir features still help lightweight classical classifiers; in the other, whether noisy energy distributions still define a useful kernel. From this perspective, the results of the present section support the claim that QML on NISQ devices should be evaluated as an end-to-end systems problem, where algorithmic design, hardware constraints, and the structure of the classical post-processing are all equally important.

6.4 Evaluation of Optimization Use-Cases

This section reports the experimental evaluation of the combinatorial-optimization case study introduced in Chapter 4.5 and detailed in Section 6.2.4, namely the BBQ-mIS workflow for graph coloring through quantum-assisted maximal Independent Set generation. In contrast with the QML use-cases discussed in Section 6.3, the objective here is not to assess predictive quality on labeled data, but to evaluate

whether the quantum component supplies branch candidates that remain useful within an exact classical optimization loop. While in Section 6.2.4 the original BBQ-mIS is validated in emulation on a benchmark of UD graphs and the full Branch-and-Bound procedure is assessed against optimal classical solutions, here we report the results of the real-hardware campaign on Aquila, where the focus shifts from full end-to-end optimality to the quality of the sampled mISs returned by the device on representative subproblems. This distinction is important because it mirrors the practical conditions under which the experiments were performed: the full workflow was first validated offline through classical emulation, whereas the hardware study was designed to test whether the sampled analog solutions retained enough combinatorial structure to support the same logic under remote-access constraints.

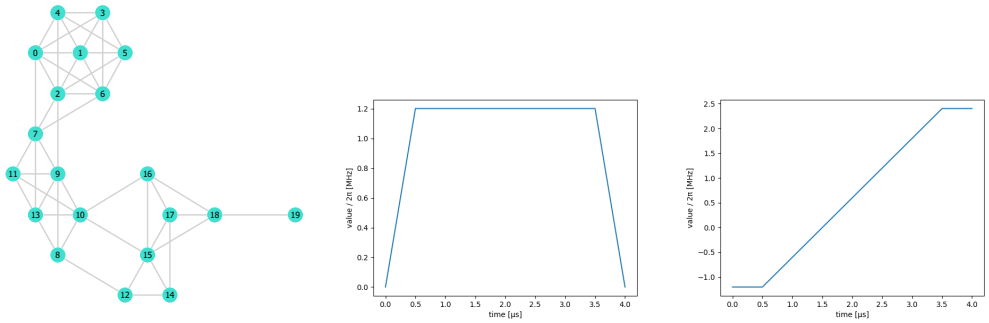
Indeed, the hardware campaign does not attempt to rerun the entire graph-coloring search online on the QPU. Because the full BBQ-mIS loop was originally developed for HPC execution and the device was available only through a remote and asynchronous cloud-like interface, it was not practical to keep the classical exact solver in a tightly synchronized loop with the QPU. Instead, the classical workflow was first executed offline to identify a representative path in the exploration tree of a 20-node graph-coloring instance derived from antenna positions in the city of Turin. Three induced subgraphs, denoted `graph_1`, `graph_2`, and `graph_3`, were then extracted from successive Branch-and-Bound nodes along that path and selected as representative MIS subproblems for hardware execution. This experimental design is essential to the thesis narrative, because it reflects the reality of early NISQ deployments: under a remote and asynchronous access model, it was not yet realistic to keep a classical exact solver blocked while waiting for every individual QPU response, so the most informative subproblems had to be identified offline and then replayed on hardware as a surrogate for the full closed-loop interaction.

Before being submitted to Aquila, each of these subgraphs had to be mapped to a hardware-compatible geometric layout. As in the graph-classification case study detailed in Section 6.2.3, this embedding phase is necessary because neutral-atom interactions are determined by spatial proximity and therefore the graph must be representable as a UD configuration under the device constraints. The methodology adopted in the real-hardware study fixes, for each selected subgraph, a specific lattice of atom positions together with the analog control waveforms defining the Hamiltonian. In particular the pulse sequence implements a QAA algorithm. The

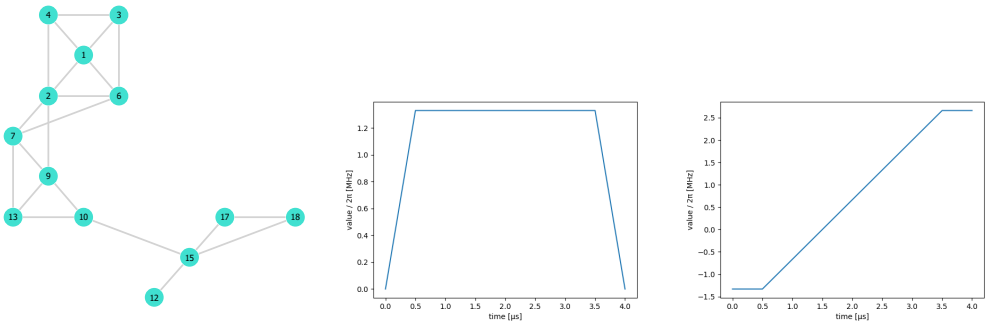
programmed controls are the Rabi frequency, the detuning, and a phase term fixed to zero. The pulse profile follows the MIS-solver logic already introduced in the methodological chapter: the Rabi amplitude rises from zero to its maximum value during the initial part of the evolution, remains constant for the central segment, and then returns to zero; the detuning starts from a negative value of the same magnitude, ramps upward, and finally remains fixed at a positive value. The exact scaling of the waveforms depends on the geometric distances in the selected lattice, because those distances determine the blockade radius and therefore the achievable interaction pattern. In the reported Aquila tasks, `graph_1` required a blockade radius of about $9.46\,\mu$m and a corresponding Rabi amplitude of approximately $1.20 \times 2\pi$ MHz, whereas the smaller residual graphs `graph_2` and `graph_3` allowed a slightly smaller blockade radius, around $9.30\,\mu$m, and a larger Rabi amplitude of approximately $1.33 \times 2\pi$ MHz. These details are visualized in Figure 6.15.

Once the three tasks are defined, the evaluation proceeds by repeated sampling on both classical emulation and real hardware. Each Aquila task is configured nominally for 1,000 shots, but the effective number of valid shots is lower because some runs fail during initialization. In the reported experiments, the numbers of retained shots are 892 for `graph_1`, 932 for `graph_2`, and 971 for `graph_3`. These valid-shot counts are then used also on the emulation side so that the two distributions can be compared on equal footing. This design choice is methodologically important: rather than comparing idealized full-sample histograms against noisy and partially failed hardware jobs, the study deliberately matches the number of usable samples, thereby isolating the effect of hardware noise on the solution distribution more faithfully.

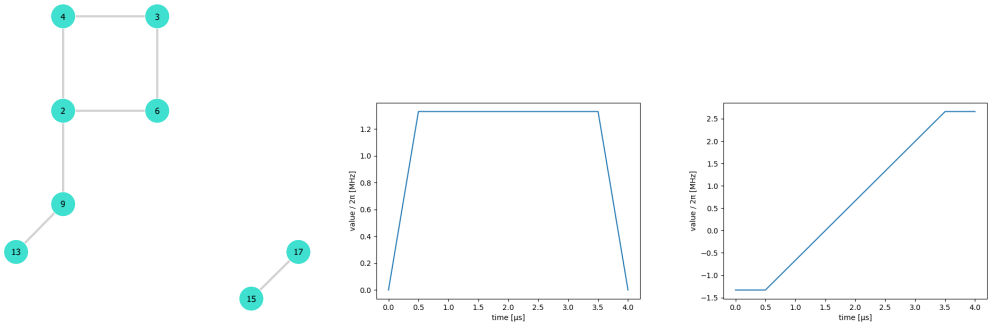
The first level of evaluation concerns the complete solution histograms. Across the three subgraphs, the source material shows that the dominant solutions observed in emulation remain visible also on Aquila, meaning that the hardware does not destroy the overall structure of the most relevant candidates. At the same time, the histograms are not identical. In hardware, the probability mass is redistributed more broadly across a larger number of bitstrings, and the ranking of the most probable configurations becomes less sharply concentrated than in emulation. This is precisely the type of deviation one would expect from decoherence, imperfect state preparation, and readout imperfections on an analog neutral-atom device. Nevertheless, from the perspective of BBQ-mIS, the important point is that the hardware histograms still retain nontrivial overlap with the meaningful combinatorial solutions found in emulation. The corresponding comparison is reported in Figure 6.16.



(a) Lattice for *graph_1*. (b) Rabi frequency for *graph_1*. (c) Detuning for *graph_1*.



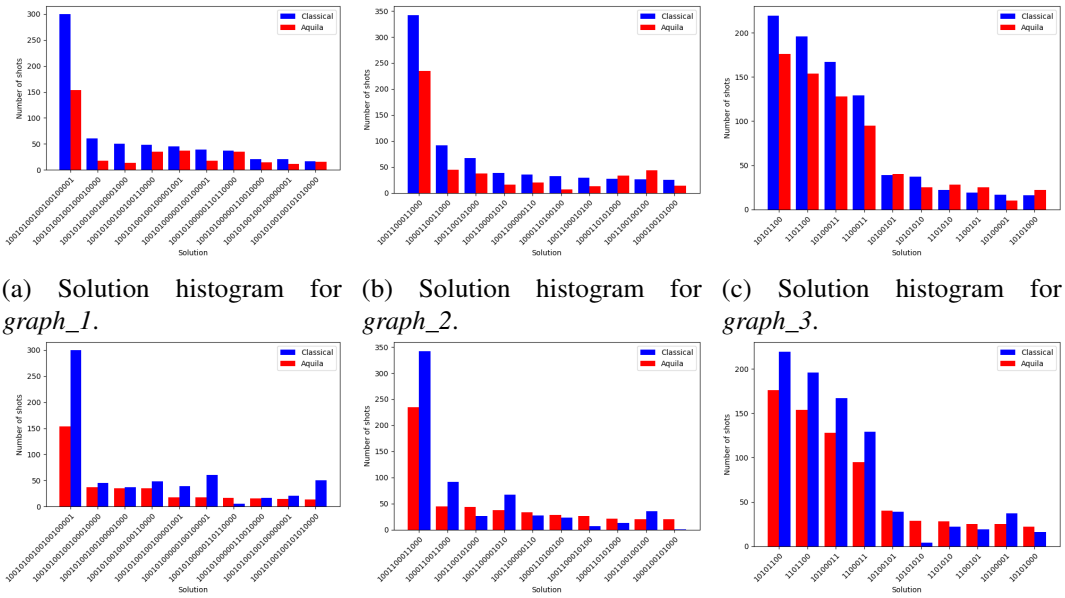
(d) Lattice for *graph_2*. (e) Rabi frequency for *graph_2*. (f) Detuning for *graph_2*.



(g) Lattice for *graph_3*. (h) Rabi frequency for *graph_3*. (i) Detuning for *graph_3*.

Fig. 6.15 Hamiltonian definition for Aquila's tasks. For each graph, the Hamiltonian is defined by the lattice, the Rabi frequency and the detuning. The phase is always kept constant at zero.

A clearer view of this effect emerges when considering the probability distributions rather than only the ranked histograms. As can be seen from Figure 6.17, The Aquila executions exhibit a broader spread of probability mass over the solution space, while the classical emulations remain more sharply concentrated on a smaller subset of configurations. This broadening is consistent across the three graphs and confirms that the hardware introduces a larger degree of stochasticity into



(a) Solution histogram for *graph_1*.

(b) Solution histogram for *graph_2*.

(c) Solution histogram for *graph_3*.

(d) Solution histogram for *graph_1*.

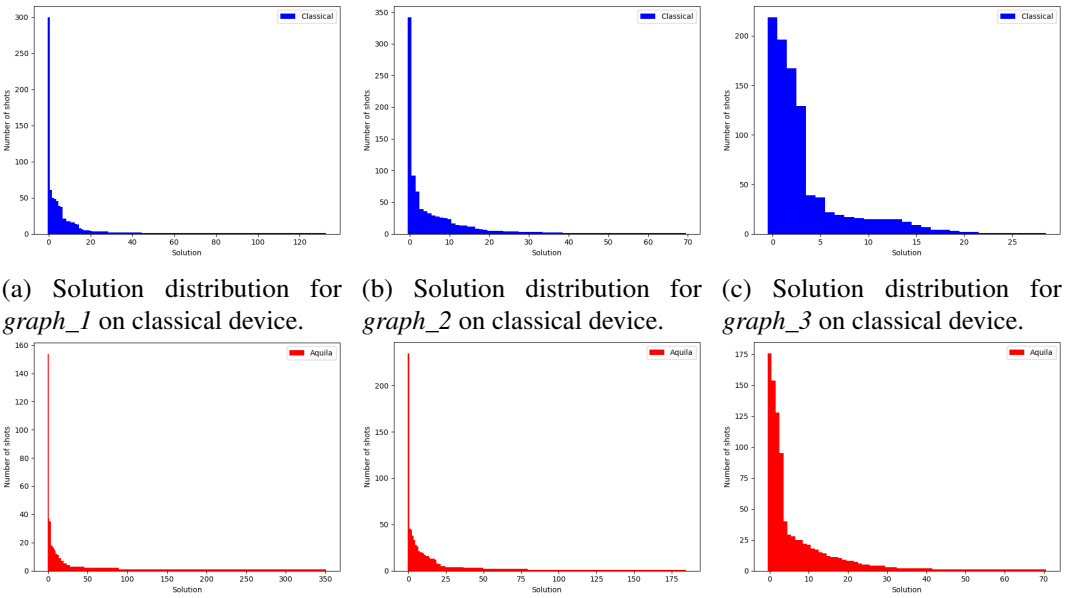
(e) Solution histogram for *graph_2*.

(f) Solution histogram for *graph_3*.

Fig. 6.16 Comparison of solution histogram between emulation on classical device and Aquila. In Figure 6.16a, 6.16b and 6.16c emulated solutions with most shots are prioritized, while in Figure 6.16d, 6.16e and 6.16f Aquila solutions with most shots are prioritized. For *graph_1* we consider a total of 892 shots, for *graph_2* 932 shots, and for *graph_3* 971 shots.

the sampling process. However, the experimental outcome should not be interpreted too negatively. In the BBQ-mIS setting, the objective is not to recover a single unique optimum with maximal probability, but rather to obtain a useful collection of mISs that can generate informative children in the Branch-and-Bound tree. From this viewpoint, a broader distribution is harmful only insofar as it diverts too much probability mass toward infeasible or combinatorially irrelevant strings. The reported results show that, although this effect is present, it does not completely erase the useful part of the distribution.

The most directly relevant metric is therefore the number of mISs that can be recovered from the samples. To evaluate this aspect more robustly, the experiments consider 128-shot batches. On the classical side, ten independent emulation runs are performed for each graph, each with 128 shots, and the mean and standard deviation of the number of recovered mISs are computed. On the hardware side, the valid Aquila shots are pooled and then partitioned into batches of the same size. The total number of mISs for each graph is computed independently through the `GenericTensorNetworks` library and used as a classical reference. This setup



(a) Solution distribution for *graph_1* on classical device.

(b) Solution distribution for *graph_2* on classical device.

(c) Solution distribution for *graph_3* on classical device.

(d) Solution distribution for *graph_1* on Aquila.

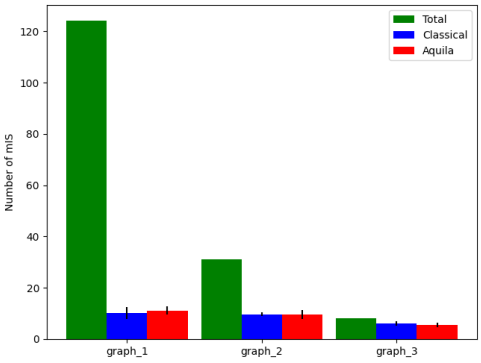
(e) Solution distribution for *graph_2* on Aquila.

(f) Solution distribution for *graph_3* on Aquila.

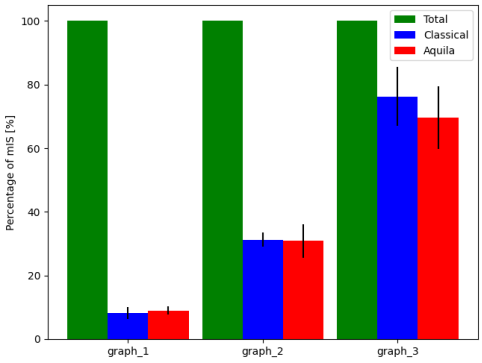
Fig. 6.17 Comparison of solution distributions (ranked probability across all sampled bitstrings) between Bloqade emulation and Aquila hardware for the three BBQ-mIS subgraphs. Top row: emulation; bottom row: Aquila. Aquila results show broader distributions consistent with decoherence, imperfect state preparation, and readout noise.

makes it possible to interpret hardware results in practical terms: the question is not whether the device can enumerate all mISs exhaustively, but whether a modest and realistic shot budget is sufficient to recover a statistically meaningful subset of them.

The corresponding comparison shows that Aquila recovers fewer mISs than classical emulation, as expected, but still returns a non-negligible number of valid candidates in every tested case. This is arguably the central empirical result of the hardware campaign. The sampled outputs are degraded by noise, yet they remain combinatorially useful enough to preserve the role of the QPU as a branch-candidate generator. In other words, the real device does not match the quality of the emulator, but neither does it collapse into producing mostly unusable strings. For the thesis narrative, this is the right level of success to emphasize: the hardware-assisted component is not shown here to outperform classical computation or to support fully online exact graph coloring, but it is shown to retain enough structural information to remain meaningful within the hybrid decomposition strategy. Figure 6.18, which reports the number of recovered mISs under 128-shot sampling, is therefore one of the most informative visual summaries for this Section.



(a) Absolute number of mISs recovered per 128-shot batch.



(b) Percentage of total mISs recovered per 128-shot batch.

Fig. 6.18 Comparison of absolute number of mISs (a) and relative percentage of the total (b) recovered from Bloqade emulation and Aquila hardware for the three BBQ-mIS subgraphs (`graph_1`, `graph_2`, `graph_3`), using 128-shot batches.

Table 6.8 summarises the hardware-execution configuration for the three representative subgraphs. The mIS recovery counts (absolute and as a fraction of the total) are reported in Fig. 6.18.

Table 6.8 BBQ-mIS hardware evaluation on Aquila: shot configuration for the three representative MIS subproblems extracted from a 20-node graph-coloring instance. Nominal budget is 1,000 shots per task; valid shots are those retained after discarding failed initializations. The 128-shot batch size is used for both the emulation (ten independent runs) and the hardware (pooled and partitioned) mIS recovery comparison.

Subgraph	Nominal shots	Valid shots (Aquila)	Initialization loss	Batch size
`graph_1`	1,000	892	10.8%	128
`graph_2`	1,000	932	6.8%	128
`graph_3`	1,000	971	2.9%	128

6.4.1 Interpretation of the optimization results

The optimization case study complements the QML results of Section 6.3 in an instructive way. In the QML studies, the key question was whether noisy quantum outputs could still support useful classification. Here, the question is whether noisy quantum outputs can still support a classical combinatorial search. The answer emerging from the experiments is cautiously positive. In emulation, BBQ-mIS is able to reproduce optimal graph-coloring results on the considered benchmark while exploring a limited number of Branch-and-Bound nodes. On real hardware, the

solution quality is no longer assessed directly through the final chromatic number, because the full online loop is not executed on Aquila; instead, the focus shifts to the quality of the sampled mISs. According to this criterion, the device remains useful, even though it produces broader and noisier distributions than emulation.

This interpretation is especially important in the broader context of the thesis. Because the quantum hardware was available only through a cloud-like and asynchronous access model, the experimentally meaningful object was not the end-to-end latency of a tightly integrated exact solver, but the quality of the candidate set returned by the remote device.The Aquila experiments therefore validate a weaker but more realistic claim: under the access conditions that characterized early NISQ experimentation, a neutral-atom QPU can act as a viable generator of maximal Independent Set candidates for a classical Branch-and-Bound workflow, even if the full hybrid loop must still be orchestrated offline or semi-offline. A final point worth stressing is that the two phases of the optimization study should be read together rather than in isolation. The emulation results establish that the BBQ-mIS decomposition is algorithmically sound and exact on the tested benchmark. The hardware results then show that, although noise reduces the quality of the sampled distributions, the real device still preserves enough of the relevant structure to make the decomposition experimentally plausible beyond emulation. This does not amount to a full practical solution of graph coloring on current hardware, and it should not be presented as such. What it does provide is a concrete proof of feasibility for one of the core ideas of this thesis: namely, that NISQ quantum hardware can already contribute usefully to larger combinatorial workflows when its role is carefully matched to the structure of the problem and to the realities of remote hybrid execution.

6.5 Performance, Scalability, and Resource Utilization

The three case studies discussed in Sections 6.3 and 6.4 make it possible to compare hybrid quantum–classical methods along a dimension that is broader than predictive accuracy or combinatorial solution quality alone, namely the way in which they consume and expose computational resources. This point is especially important in the context of the present thesis, because the experimental campaign was conducted

under remote-service conditions, with quantum hardware accessed through managed interfaces rather than as a tightly integrated local accelerator. As a consequence, the relevant notion of performance is not limited to raw device speed, but must also include the structure of the workload, the amount of classical preprocessing and post-processing required, the dependence on shot budgets, and the scalability bottlenecks induced by orchestration and access constraints.

A first common observation is that, across all three use-cases, classical computation remains an indispensable and often dominant component of the end-to-end pipeline. In the QRC study, classical preprocessing, resampling, and downstream classification account for a substantial part of the execution flow, while the quantum device is used specifically as a feature generator. In the QEK study, hardware-compatible graph embedding, energy-distribution construction, and SVM training all remain classical, even though the kernel itself is induced by quantum evolution. In the BBQ-mIS study, the overall graph-coloring logic, including bounds, pruning, and tree management, is entirely classical, while the quantum device acts as a specialized generator of maximal independent set candidates. This confirms one of the central claims of the thesis: in the NISQ regime, the practical computational object is the hybrid stack as a whole, rather than the isolated QPU kernel.

In the QRC case, the performance profile is shaped by the fact that the quantum routine is invoked independently for each data sample and for multiple timesteps, producing an expanded feature space that is then processed by lightweight classical models. From a scalability viewpoint, this creates a workload whose quantum cost grows essentially with the number of samples, while the classical training phase remains relatively inexpensive compared with deeper nonlinear alternatives such as the DNN benchmark. This asymmetry is visible in the training-time comparison reported in Table 6.3, where the readout models remain much cheaper to train than the deep baseline even when resampling is applied. At the same time, the hardware campaign had to be restricted from `CARDS_30000` to the smaller `CARDS_2500` and `CARDS_1000` subsets, which shows that the main practical scalability limitation of the QRC workflow lies not in the classical learner, but in the cost and availability of quantum feature generation under finite shot budgets. In this sense, QRC exhibits a favorable classical training profile but a sample-driven quantum acquisition cost.

The QEK case follows a different scaling logic. Here, the quantum device is not used to generate a feature vector for each sample in isolation, but to acquire

graph-dependent energy distributions from which a kernel matrix is later assembled classically. The direct hardware cost is therefore concentrated in per-graph measurement acquisition, while the subsequent similarity computation scales classically with the size of the dataset. This structure is advantageous because the expensive quantum stage does not itself scale quadratically with the number of samples, but it still implies a nontrivial classical bottleneck in the assembly and use of the full kernel matrix. The methodological choice of optimizing pulse parameters on `PROTEINS12` and then transferring them to `PROTEINS256` is therefore also a scalability strategy: it confines the most expensive design-space exploration to a classically tractable subset while preserving the possibility of running a larger benchmark on hardware. The shot-subsampling analysis reported in Table 6.7 further reinforces this interpretation, since it shows that competitive classification can still be obtained with moderate shot counts, thereby reducing the practical cost of hardware execution.

The optimization case study displays yet another scaling profile. In BBQ-mIS, the quantum resource is neither used in a dense sample-wise fashion as in QRC nor in a dataset-wide similarity construction as in QEK. Instead, it is invoked irregularly on state-dependent MIS subproblems arising from the evolution of a classical Branch-and-Bound search. This means that the main scalability issue is not only the cost of each quantum execution, but also the impossibility of predicting *a priori* when and how many quantum calls will be needed during the search. On the emulation side, the method remains tractable on the selected benchmark, requiring at most 20 explored nodes per instance and approximately 8000 core-hours for the overall campaign. On the hardware side, however, the experimental question had to be reformulated more narrowly, focusing on selected representative subproblems rather than on a fully online end-to-end execution. This makes clear that the scalability bottleneck of the optimization workflow lies as much in irregular orchestration as in raw quantum cost.

A second cross-case observation concerns the relation between noise and resource efficiency. In the QRC study, hardware noise reduces feature fidelity and therefore decreases the information extracted per quantum execution, especially on the smallest dataset. In this setting, poor feature quality effectively lowers the return on each shot budget, since more hardware effort is spent to produce less discriminative features. In the QEK study, by contrast, the induced distributions remain sufficiently structured that the same order of shot budgets continues to support competitive classification performance. In the BBQ-mIS study, the relevant resource-efficiency question is

whether a fixed number of shots yields enough valid maximal independent sets to support the classical search. The answer is positive, though with a clear degradation relative to emulation. These three behaviors show that "resource utilization" in hybrid quantum–classical applications should not be interpreted only as hardware occupancy or number of submitted shots, but also as the amount of task-relevant information extracted from those shots.

A third point concerns the role of scalability through classical mitigation. In both QML studies, larger datasets improve robustness, but they do so in different ways. In QRC, increasing the number of samples gives the downstream classical learner more statistical leverage to compensate for noisy quantum features. In QEK, increasing the number of graphs improves the stability of the kernel-based classification problem and makes the comparison with the classical baseline more meaningful. In the optimization study, scalability does not manifest primarily through larger datasets on hardware, but through the ability of the classical Branch-and-Bound layer to absorb noisy quantum outputs using feasibility checks, maximality checks, and pruning rules. In all three cases, scalability is therefore not simply a matter of accommodating larger quantum programs; it also depends on whether the surrounding classical structure can absorb imperfections without collapsing the utility of the overall workflow.

From the standpoint of resource utilization, the Chapter also suggests a broader interpretation that connects naturally with the execution models of Chapter 5. The three case studies stress the underlying platform in different ways. QRC resembles a repeated feature-extraction workload with many conceptually independent per-sample quantum invocations. QEK resembles a kernel-acquisition workload in which the expensive quantum stage is followed by a classically costly but highly structured aggregation step. BBQ-mIS resembles an irregular decomposition workload in which quantum invocations are sparse, heterogeneous, and tightly coupled to classical search state. This distinction is important because it explains why there is no single best execution strategy for all hybrid methods. Different application structures expose different pressure points in terms of batching, queue sensitivity, synchronization, and useful work extracted per hardware call.

Overall, the evidence collected in this Chapter supports a resource-oriented interpretation of NISQ hybrid computing. Performance is not determined only by the quality of the quantum output, but also by the extent to which the workflow confines the quantum device to a role that is information-rich, structurally narrow,

and efficiently exploitable by the classical layers. Scalability is not determined only by qubit count or circuit depth, but also by whether the surrounding preprocessing, orchestration, and post-processing stages remain manageable as the workload grows. Finally, resource utilization is not captured adequately by device occupancy alone, but must be understood in terms of end-to-end usefulness of the hybrid execution under realistic access constraints.

Table 6.9 consolidates these observations into a compact cross-case comparison, aligning workload patterns, quantum component roles, dominant bottlenecks, and key findings for all three studies.

6.6 Discussion of Results

The results reported in this Chapter do not support a simplistic conclusion according to which hybrid quantum–classical methods are either uniformly promising or uniformly ineffective in the NISQ regime. Rather, they show a more nuanced picture in which the experimental value of a quantum component depends strongly on the role assigned to it within the larger workflow, on the way noise propagates through the downstream classical stages, and on the practical access conditions under which the hardware is used. This is precisely the perspective advocated throughout the thesis: the meaning of "performance" in the NISQ era is fundamentally contextual and cannot be inferred from quantum subroutines in isolation.

A first major outcome is that the three case studies exhibit different degrees and forms of robustness to realistic hardware imperfections. In the QRC study, the quantum reservoir remains competitive in classical emulation, but the move to real hardware induces a visible degradation, especially on smaller datasets and for the position-based encoding. This indicates that, when the quantum contribution is directly materialized as a feature vector consumed by a downstream classifier, distortions in the hardware dynamics can propagate rather transparently to the final task. In the QEK study, the situation is more favorable: the hardware-induced perturbation of the energy distributions does not destroy the usefulness of the resulting kernel and, on the smaller benchmark, may even improve it slightly. In the BBQ-mIS study, the real device yields noisier and broader distributions than emulation, but still preserves enough structure to produce combinatorially useful maximal independent sets. The common lesson is therefore not that noise is harmless, but that its practical

impact depends on whether the quantum output is used as a raw representation, as a similarity descriptor, or as a generator of candidate solutions.

A second outcome is that no broad quantum advantage can be claimed from the present evidence, yet several narrower forms of practical relevance do emerge. For QRC, the key positive result is that a fixed quantum reservoir can produce informative nonlinear features while shifting the trainable burden to lightweight classical models. For QEK, the main result is that a hardware-native graph kernel remains competitive with a strong classical baseline on a benchmark of embedded protein graphs. For BBQ-mIS, the relevant conclusion is not end-to-end superiority over exact classical graph coloring, but the feasibility of using a neutral-atom device as a source of meaningful subproblem candidates within a larger exact search logic. These are modest but significant claims, and they are more scientifically defensible than generic assertions of near-term quantum advantage.

A third point concerns the importance of classical scaffolding. Across all three studies, the quantum routine is useful only because it is embedded within a carefully designed classical envelope. In QRC, preprocessing, resampling, and classical readout design are crucial to the final result. In QEK, graph embedding, pulse selection, histogram processing, and SVM training are all essential components of the method. In BBQ-mIS, graph-theoretic bounds, pruning strategies, and branch ordering are what turn quantum samples into a coherent optimization workflow. This means that the experiments should not be interpreted as comparisons between "quantum" and "classical" in a strict dichotomous sense. Instead, they demonstrate that useful behavior arises from an appropriate division of labor between the two domains. This confirms that algorithm design and platform-aware workflow design are inseparable in practice.

A fourth outcome, perhaps the most important from the viewpoint of this dissertation, is the decisive role played by the hardware access model. As already underlined, at the time of the experimental campaign, there were no production-grade hybrid clusters in which the target QPUs could be treated as locally co-scheduled accelerators. The quantum devices were instead available only through cloud-like remote interfaces, with asynchronous submission, finite shot budgets, queueing uncertainty, and occasional initialization failures. This condition is not a peripheral implementation detail; it changes the very meaning of what can be evaluated experimentally. In the present Chapter, it explains why extensive hyperparameter tuning had to be

performed offline, why hardware subsets had to be smaller than classical emulation sets, and why the optimization case study could only assess selected subproblems rather than a fully synchronized online Branch-and-Bound loop. Read in this way, the results are not weakened by the remote-access condition; rather, they become representative of the realistic operational regime in which early NISQ applications were actually executed.

This also clarifies how the present experimental Chapter relates to the discussion on infrastructure of Chapter 5. The application studies of this Chapter are not direct performance benchmarks of vQPUs, workflow decomposition, or malleability. Nevertheless, they reveal exactly why such execution strategies matter. The QRC case exposes a repeated, sample-driven quantum invocation pattern. The QEK case exposes a structured but potentially large measurement-acquisition workflow. The BBQ-mIS case exposes irregular and state-dependent quantum calls. These patterns correspond closely to the workload families analyzed earlier from a scheduler and runtime perspective. Therefore, the value of the present results is twofold: they validate the application-level feasibility of the proposed methods, and they also ground the systems-level arguments of the thesis in concrete, experimentally observed workload behaviors.

A final point concerns the scope of the conclusions that can legitimately be drawn. The Chapter shows that analog neutral-atom platforms can support meaningful hybrid workloads in both QML and combinatorial optimization, but always under carefully delimited roles and scales. It does not show that such devices already provide a general-purpose replacement for classical methods. It does not show that end-to-end quantum acceleration is presently available under realistic operational constraints. What it does show is more specific and, in some sense, more important for the NISQ era: when the quantum device is assigned a role that is structurally compatible with its native dynamics and when the surrounding classical workflow is designed to absorb its limitations, the resulting hybrid method can remain experimentally useful even in the presence of noise, remote access, and limited resources. This is the main empirical conclusion of the Chapter.

In this sense, Chapter 6 closes the experimental part of the dissertation with a deliberately balanced message. The evidence does not justify exaggerated claims of imminent practical quantum supremacy in the application domains considered here. At the same time, it does justify a more grounded and constructive conclusion: the

NISQ regime already admits nontrivial forms of hybrid usefulness, provided that one abandons monolithic views of quantum acceleration and instead embraces a cross-layer perspective in which algorithms, runtimes, resource models, and hardware access policies are optimized together. This broader interpretation naturally motivates the discussion of Chapter 7, where the lessons of the individual case studies are lifted to a more general set of guidelines for the design of future hybrid QC–HPC systems.

Table 6.9 Cross-case-study summary: workload patterns, quantum component roles, dominant bottlenecks, and key empirical findings across the three hybrid quantum–classical use-cases evaluated in this Chapter. The bottleneck and finding columns reflect conditions at the time of the experimental campaign (remote cloud-like QPU access, finite shot budgets, asynchronous submission).

Case study	**Workload pattern**	**Quantum role**	**Dominant bottleneck**	**Key finding**
QRC (credit-card default)	Dense per-sample invocations; one quantum execution per sample; repeated over all dataset samples	Fixed nonlinear feature generator (reservoir); trainable burden confined to classical readout	Shot budget $\times$ dataset size; hardware noise reduces feature fidelity, especially on small datasets	Competitive with DNN in emulation; hardware degradation partially absorbed by downstream classifier at larger data scales
QEK (graph classif.)	Per-graph energy-distribution acquisition; linear scaling in graph count; kernel matrix assembled classically	Pairwise similarity descriptor via hardware-native Rydberg-dynamics energy distributions	Graph embedding feasibility (UD constraint); shot budget for histogram estimation; $O(N^2)$ kernel-matrix assembly	Matches SPK on `PROTEINS256` (F1: 65.6% vs. 65.3%); hardware noise non-detrimental on `PROTEINS12`; competitive under 100-shot budgets
BBQ-mIS (graph coloring)	Irregular, state-dependent calls at B&B tree nodes; sparse, bursty, and heterogeneous invocations	Combinatorial candidate generator (mIS proposals) inside a classical Branch-and-Bound framework	Remote access latency; orchestration irregularity; fully online closed loop infeasible under asynchronous cloud-like access	Optimal colorings in emulation; Aquila returns fewer but non-trivial mIS candidates; hardware viable as offline branch-candidate generator

Chapter 7

Discussion

This dissertation started from the observation that, in the NISQ era, hybrid quantum–classical computation cannot be understood adequately by focusing on algorithms alone. The practical behavior of a quantum-enhanced application is shaped not only by the mathematical structure of the method, but also by the way it is encoded, executed, orchestrated, and exposed through real hardware and software stacks. The analyses and experiments reported in the previous Chapters confirm this view consistently across both application-oriented and infrastructure-oriented dimensions.

On the application side, the thesis has examined Quantum Machine Learning and combinatorial optimization under realistic NISQ constraints, showing that the usefulness of a quantum component depends strongly on the role assigned to it within a broader classical workflow. On the systems side, the thesis has shown that the efficiency of hybrid execution is heavily influenced by resource asymmetry, access policies, latency, queueing, and the granularity at which classical and quantum phases can be coordinated. Considered together, these results support the central thesis of the dissertation: meaningful improvements in the NISQ regime emerge not from isolated quantum kernels, but from cross-layer co-design across the algorithmic, runtime, and platform levels.

The present Chapter discusses the implications of these findings. Section 7.1 revisits the main lessons from the QML and optimization use-cases. Section 7.2 shifts the focus to the infrastructure integration problem and summarizes what the infrastructure-level analyses imply for hybrid execution. Section 7.3 draws from these observations to outline a practical path toward production-grade hybrid QC–

HPC systems. Finally, Section 7.4 reflects on how the conclusions of this work may evolve beyond the current NISQ setting.

7.1 Lessons from QML and Optimization Use-Cases

A first lesson emerging from the application studies is that different classes of hybrid quantum algorithms stress the platform in fundamentally different ways. The QRC case study is representative of a feature-generation workflow in which the quantum device is invoked repeatedly on individual samples and the resulting representation is consumed by lightweight classical learners. The QEK case study instead follows a distribution-based similarity construction pattern, where the quantum stage induces graph-dependent empirical distributions that are then aggregated into a classical kernel matrix. The BBQ-mIS case study represents an irregular decomposition workflow, in which quantum subroutines are called selectively within a larger classical search procedure. These distinctions are not incidental; they determine which bottlenecks dominate, how noise propagates, and what kind of resource-management strategies are likely to be effective.

A second lesson is that, in the NISQ regime, the most promising role for a quantum device is often narrow but structurally meaningful rather than broad and monolithic. In the QRC workflow, the quantum processor acts as a nonlinear feature transformer; in QEK, it acts as a generator of graph-dependent distributions that define a kernel; in BBQ-mIS, it acts as a source of candidate maximal independent sets for a classical Branch-and-Bound procedure. In none of these cases is the quantum component responsible for solving the whole application in isolation. Instead, it contributes a specialized transformation or sampling capability that is subsequently amplified, filtered, or stabilized by classical processing. This pattern appears repeatedly throughout the dissertation and should be regarded as one of the defining characteristics of practical NISQ application design.

A third lesson concerns the interaction between noise and application structure. The experiments do not support a uniform view according to which hardware noise is either uniformly fatal or uniformly benign. Rather, its effect depends on the semantic role of the quantum output. In QRC, where the output is a direct feature vector, hardware noise degrades the learned representation rather transparently, especially in limited-data regimes. In QEK, where the relevant object is a distribution-induced

similarity measure, hardware perturbations are partially absorbed by the kernel construction and do not necessarily harm final classification. In BBQ-mIS, where the output is a sampled set of candidate combinatorial structures, the crucial question is not fidelity to emulation in a strict state-level sense, but whether enough feasible and useful candidates survive after classical filtering. This distinction is important because it suggests that noise robustness should be analyzed at the level of end-to-end task semantics, not only at the level of low-level device observables.

A fourth lesson is that classical scaffolding is not a secondary implementation layer but an essential part of the method. In the QRC study, preprocessing, dimensionality reduction, clustering of features, resampling, and classifier selection are all decisive components of the final workflow. In QEK, graph filtering, hardware-aware embedding, pulse selection, histogram construction, and SVM training are equally central. In BBQ-mIS, the quantum stage would not be useful without graph-theoretic bounds, node prioritization, pruning rules, and state management in the classical Branch-and-Bound layer. Therefore, it would be misleading to interpret the experiments as direct comparisons between a purely quantum method and a purely classical one. The practical object of interest is always a hybrid pipeline in which the classical component is both a support structure and an active source of robustness, scalability, and interpretability.

A fifth lesson is that larger-scale usefulness in the NISQ era often comes from classical absorption of quantum imperfections rather than from pristine quantum execution. This is visible in the QRC study, where increasing the dataset size partially mitigates the impact of noisy reservoir features because the downstream learner gains more statistical leverage. A related phenomenon appears in the QEK study, where the larger dataset allows a more stable evaluation of the induced kernel and a more meaningful comparison with the classical baseline. In the optimization case, the classical search layer absorbs noisy quantum outputs by converting them into branch candidates and then filtering them according to feasibility and maximality conditions. In all three cases, scalability is not simply a matter of executing larger quantum programs; it is also a matter of embedding the quantum routine inside a classical structure that can compensate for its limitations.

A final lesson from the application side is methodological. The experiments support carefully delimited claims of feasibility and usefulness, but not strong claims of broad near-term quantum advantage. The QRC case shows that quantum reservoirs

can produce competitive representations in idealized settings and still retain partial usefulness on hardware. The QEK case shows that a hardware-native graph kernel can remain competitive with a strong classical baseline on an embedded dataset. The BBQ-mIS case shows that a real neutral-atom device can already contribute useful subproblem candidates within a larger combinatorial workflow. These are meaningful results, but their significance lies precisely in their specificity. They suggest that the most scientifically responsible way to assess NISQ applications is through narrow, well-contextualized claims tied to realistic execution conditions rather than through generic narratives of imminent disruption.

7.2 Lessons from Cross-Layer Integration

If the application studies clarify what hybrid quantum methods can do, the infrastructure studies clarify under what conditions they can do it efficiently. The most important lesson from the infrastructure analysis is that hybrid QC–HPC execution is structurally different from traditional HPC execution. Classical HPC systems are optimized for long-running, largely homogeneous workloads with relatively stable resource demands. Hybrid quantum workloads, by contrast, are interaction-intensive, latency-sensitive, and often highly asymmetric in their use of classical and quantum resources. This mismatch means that straightforward extensions of traditional scheduling and allocation policies are rarely sufficient.

A recurring source of inefficiency is the extreme asymmetry between abundant classical resources and scarce quantum resources. In a typical hybrid setting, many CPU or GPU cores may be available, whereas only one or a few QPUs can be exposed to users. This asymmetry turns the quantum device into a bottleneck and makes naive co-allocation wasteful: classical resources may remain reserved while a job is waiting for quantum access, or the QPU may remain underutilized because users are granted overly rigid exclusive access. Chapters 5 and 6 show that this issue is not a marginal engineering detail but one of the defining constraints of current hybrid computing environments.

A second lesson is that there is no single optimal execution strategy for all hybrid workloads. The vQPU model is most effective when quantum phases are short relative to classical phases and when multiple jobs can share the same quantum resource through time-based multiplexing with limited user-visible overhead. Work-

flow-based decomposition is effective when the structure of the application can be made explicit and when different phases can be submitted with separate resource requirements, thereby avoiding idle reservations. Malleability becomes particularly valuable when the application can dynamically release and reacquire classical resources in response to changing computational phases. These three strategies address the same underlying problem, namely the mismatch between static reservations and dynamic hybrid workloads, but they do so at different abstraction levels and with different assumptions about programmability, transparency, and runtime support.

A third lesson concerns transparency versus explicitness in programming models. The vQPU approach is attractive because it is largely transparent to the user application: jobs can benefit from better quantum sharing without deep changes to application structure. Workflow decomposition, by contrast, requires the application to be exposed explicitly as a sequence or graph of heterogeneous steps, which increases the burden on the developer but also creates more opportunities for scheduler-aware optimization. Malleability occupies an intermediate point: it preserves the identity of the application as a single running program, but requires code-level adaptation to support dynamic resizing and state management. This design space is important because it shows that the question of efficient hybrid execution is not purely about scheduler policy; it is also about which responsibilities are assigned to the user, to middleware, and to the resource manager.

A fourth lesson is that observability and workload characterization are indispensable. It is impossible to manage hybrid resources efficiently without understanding the time scales of classical and quantum phases, the distribution of quantum invocation patterns, and the variability induced by queueing and backend-side behavior. The analyses of Chapter 5 show that metrics such as quantum occupancy, mean queue time, makespan, and node usage are not ancillary reporting tools; they are the basis for identifying which integration strategy is appropriate for a given workload class. In other words, practical hybrid integration requires not only mechanisms for scheduling and orchestration, but also mechanisms for measuring and modeling the behavior of hybrid jobs at runtime.

A fifth lesson concerns the role of real deployment. The integration of an on-premises quantum system and the analysis of cloud-like access to neutral-atom hardware both show that platform constraints feed back directly into algorithmic and middleware design. Queueing policies, asynchronous completion semantics,

finite shot budgets, and backend-specific submission interfaces all shape the effective programming model seen by the user. This confirms that algorithm–platform integration cannot be reduced to a clean layering in which infrastructure merely executes decisions made elsewhere. Instead, infrastructure constraints actively define the space of feasible algorithmic choices. This infrastructure-to-algorithm feedback loop is one of the central conceptual outcomes of the dissertation.

Taken together, these observations suggest that hybrid QC–HPC integration should be treated as a systems problem with strong algorithmic consequences. The role of the platform is not just to provide access to a QPU, but to shape the timing, granularity, and coordination of the entire hybrid execution. Symmetrically, the role of the application designer is not just to formulate a quantum-enhanced method, but to expose enough structure for the runtime and scheduler to act intelligently. This mutual dependence is the practical meaning of the cross-layer viewpoint adopted throughout the thesis.

7.3 Toward Production-Grade Hybrid QC–HPC Systems

The results of this dissertation suggest that production-grade hybrid QC–HPC systems will require more than incremental adaptation of existing HPC and quantum software stacks. They will need a more explicit co-design of interfaces, execution models, and resource-management policies, with the aim of making hybrid applications both efficient and operationally manageable. Although current NISQ systems are still far from widespread production deployment, the findings of this work already point to a number of concrete design principles.

A first requirement is the explicit treatment of quantum resources as schedulable shared infrastructure rather than as ad hoc external services. Whether access is mediated through local integration, vQPU-style virtualization, cloud interfaces, or workflow engines, the QPU must be represented inside the system control plane in a way that allows the scheduler and runtime to reason about it. Without such representation, quantum invocations remain opaque external events, and the rest of the platform cannot optimize classical reservations around them. This is one of the main motivations behind the abstraction work developed in Chapter 5.

A second requirement is support for multiple integration modes. The case studies and execution analyses show that different workloads demand different orchestration patterns. Some applications benefit from transparent virtualization, some from explicit workflow decomposition, and others from dynamic resizing or on-the-fly reservation mechanisms. A production-grade system should therefore not enforce a single integration model universally. Instead, it should expose a family of interoperable abstractions, allowing the same hardware substrate to support both transparent and application-aware scheduling modes. This flexibility is especially important in the NISQ era, where workload heterogeneity is high and stable best practices have not yet converged.

A third requirement is the minimization of idle reservations. Hybrid applications frequently alternate between phases that use different resource types, and the experiments clearly show that static co-allocation can lead to substantial waste. Production-grade systems should therefore provide mechanisms for phase-aware scheduling, either by decomposing workflows explicitly or by allowing applications to release and reacquire resources dynamically. The specific implementation may vary, but the principle remains the same: scarce resources must not be blocked simply because the software stack lacks the means to represent changing phase requirements.

A fourth requirement is clear separation of concerns between application logic, runtime control, and scheduling policy. One of the practical difficulties in current hybrid systems is that orchestration decisions are often embedded directly into application code or handled in an ad hoc manner by bespoke scripts. This hinders portability, reproducibility, and fair sharing of the infrastructure. A more robust path is to maintain a clean conceptual separation: the application should expose its structure and resource-relevant behavior; the runtime should manage control flow, submission, and monitoring; and the scheduler should arbitrate resource access according to platform-wide objectives such as utilization, fairness, and predictability. This separation does not eliminate cross-layer interaction, but it makes that interaction explicit and manageable.

A fifth requirement is richer observability and predictive control. Production systems will need mechanisms to monitor hybrid jobs at a finer granularity than traditional batch workloads, including metrics related to quantum invocation frequency, queue exposure, shot consumption, and phase imbalance. Such observability is needed not only for performance tuning but also for fairness and accounting.

If quantum resources remain scarce and expensive, users and operators will need meaningful models of how hybrid jobs consume them and what return they provide at the application level.

A sixth requirement is reproducibility under imperfect and heterogeneous access conditions. The experiments of this thesis were shaped heavily by the fact that quantum devices were often available only through remote and cloud-like interfaces. This condition is likely to persist for some time, even as local quantum accelerators become more common in selected centers. Production-grade systems should therefore be designed to accommodate a spectrum of access modes, from local accelerator-style integration to remote managed services. In each case, the runtime and middleware should preserve enough metadata and execution context to make comparative experimentation and debugging possible.

Beyond these requirements, the results of the thesis also suggest a broader strategic point. The path to production-grade hybrid computing is unlikely to come from a single decisive breakthrough, either in hardware performance or in software orchestration. More realistically, it will emerge from gradual convergence between several layers: hardware that becomes more stable and programmable, algorithms that expose narrower and more meaningful quantum roles, middleware that can coordinate heterogeneous phases, and schedulers that can reason about dynamic and asymmetric workloads. The cross-layer approach advocated here is therefore not only an analytical framework for the present; it is also a practical roadmap for the engineering of future hybrid systems.

7.4 Outlook Beyond the NISQ Era

Although this dissertation is explicitly situated in the NISQ regime, many of its conclusions remain relevant beyond it. What is likely to change in the future is not the need for hybrid orchestration itself, but the balance of concerns across the stack. As quantum hardware improves, some bottlenecks that dominate current systems may weaken, while others may become more salient.

A first likely evolution concerns the role of noise. In the present work, noise is a central determinant of application behavior, shaping both algorithmic design and experimental interpretation. In a future regime with better error rates, longer

coherence times, and eventually logical qubits, some of the current limitations on circuit depth and sampling quality may recede. This could enlarge the class of viable quantum subroutines and reduce the need for certain forms of classical compensation. However, this would not remove the systems problem. Deeper quantum programs, larger problem instances, and more elaborate hybrid loops may actually increase the importance of scheduling, orchestration, and efficient hardware sharing.

A second likely evolution concerns hardware diversity. The thesis has considered mainly neutral-atom analog platforms and gate-based systems in the context of virtualization and integration. Future hybrid infrastructures are likely to involve a richer mix of modalities, including superconducting processors, neutral atoms, trapped ions, photonic platforms, annealers, and perhaps early fault-tolerant co-processors. This heterogeneity will make abstraction and portability more important, not less. Full-stack methods will need to cope with multiple programming paradigms, multiple time scales, and different cost models for access and compilation.

A third likely evolution concerns the granularity of integration with HPC systems. In the current NISQ period, many experiments still rely on cloud-like or loosely coupled access, and this strongly conditions the design of hybrid workflows. In a more mature stage, tighter accelerator-style coupling may become feasible in selected environments. This would reduce some forms of latency and enable finer-grained interaction patterns, but it would also increase the demands placed on resource managers, accounting systems, and application runtimes. In other words, closer integration does not eliminate the cross-layer challenge; it changes its operational form.

A fourth likely evolution concerns the type of application claims that become credible. In the present regime, carefully delimited feasibility claims and narrowly scoped usefulness claims are the most scientifically responsible outcomes. Beyond NISQ, one may expect stronger performance claims to become possible in selected domains. Yet even then, the lesson of this dissertation should remain valid: any meaningful claim of advantage will need to be evaluated in context, against realistic baselines, and under realistic resource-access conditions. End-to-end usefulness will still depend on the interplay between algorithmic quality, systems efficiency, and operational constraints.

Finally, the outlook beyond the NISQ era reinforces the broader intellectual position of the thesis. Quantum computing should not be viewed only as a new

kind of processor, nor only as a source of new algorithms, but as a new layer in heterogeneous computing systems whose value emerges through coordinated interaction with the rest of the stack. In the current regime, this coordination is necessary to compensate for limitations; in future regimes, it will be necessary to exploit expanded capabilities efficiently. For this reason, the cross-layer perspective developed in this dissertation should be understood not as a temporary workaround for immature hardware, but as a durable framework for reasoning about quantum computing as an integral component of future high-performance and data-intensive infrastructures.

From this perspective, the NISQ era should be seen less as a disappointing prelude to fault tolerance and more as the phase in which the practical disciplines of hybrid quantum computing are being established. The main contribution of this thesis lies precisely in that space: it shows, through concrete case studies and systems-level analyses, that the path toward useful quantum computing is already shaped by questions of orchestration, abstraction, and platform design. Those questions will not disappear when hardware improves. They will simply become the basis on which larger and more powerful quantum-enabled computing ecosystems are built.

Chapter 8

Conclusions

This dissertation has investigated hybrid quantum–classical computation in the NISQ era from a cross-layer perspective, spanning algorithmic design, workload characterization, resource management, and platform-level integration. The central argument, articulated in Chapter 1 and substantiated throughout the thesis, is that useful quantum-enhanced computation in the NISQ regime does not arise from isolated algorithmic improvements alone. It emerges from coordinated co-design across abstraction layers: algorithms must be matched to the structural capabilities of the hardware; execution models must account for the latency and asymmetry of hybrid workflows; and resource management strategies must reflect the operational realities of shared, heterogeneous HPC–QC systems. The following sections revisit each research question with the evidence gathered, summarize the main contributions alongside their empirical findings, consolidate the principal limitations of the work, and outline concrete directions for future research.

8.1 Answers to the Research Questions

RQ1: How can hybrid quantum–classical algorithms for QML and combinatorial optimization be designed and implemented to operate effectively under NISQ constraints? Chapters 3 and 4 address this question through three complementary case studies. In QML, both Quantum Reservoir Computing and the Quantum Evolution Kernel exploit hardware-native analog dynamics rather than deep parameterized circuits, thereby sidestepping the trainability and barren-plateau

issues that afflict variational approaches on NISQ devices. In combinatorial optimization, the BBQ-mIS method replaces the full-stack execution of a quantum solver with a carefully scoped use of a neutral-atom device as a maximal Independent Set generator within a classical Branch-and-Bound loop. Across all three studies, the experiments confirm a consistent design principle: the most effective role for a NISQ quantum component is narrow but structurally meaningful, a nonlinear feature transformer in QRC, a distribution-induced similarity measure in QEK, a combinatorial candidate generator in BBQ-mIS, rather than a standalone solver responsible for the entire application. Experimental results in Chapter 6 support feasibility under realistic conditions and caution against broad claims of quantum advantage.

RQ2: What execution models best capture the behavior of hybrid quantum–classical workloads in HPC environments? Chapter 5 addresses this question through a taxonomy of workload patterns and a formal analysis of their resource interaction profiles. Three structurally distinct patterns emerge: dense per-sample feature generation (QRC), per-graph kernel acquisition (QEK), and irregular state-dependent decomposition (BBQ-mIS). Each pattern induces a different bottleneck structure: QRC is dominated by shot budget times dataset size; QEK by graph embedding feasibility and quadratic kernel assembly; BBQ-mIS by orchestration irregularity and the impossibility of closed-loop online execution under asynchronous remote access. The analysis in Section 5.1 formalizes the time-scale imbalance between classical and quantum phases and identifies the conditions under which naive co-allocation wastes resources. This characterization provides the conceptual basis for the three platform-aware strategies evaluated in the same Chapter.

RQ3: How can scarce quantum resources be efficiently integrated and virtualized within HPC infrastructures? The vQPU abstraction, introduced in Section 5.3, addresses this question by exposing quantum hardware as a schedulable shared resource within an HPC control plane. The design enables time-based multiplexing of a single physical QPU across multiple concurrent hybrid jobs, without requiring modifications to user application code. The performance characterization reported in Sections 5.4 and 5.4.2 demonstrates that vQPU-based sharing reduces QPU idle time and improves overall system throughput in multi-tenant settings, at the cost of a modest and predictable per-job overhead. Integration with SLURM is achieved through a dedicated scheduling plugin that represents the quantum re-

source as a generic consumable, preserving compatibility with standard HPC batch-submission workflows.

RQ4: Which platform-aware resource management strategies improve the performance and scalability of hybrid QC–HPC workloads? Section 5.5 and the complementary analyses of workflow decomposition (Section 5.5.1) and malleability (Section 5.5.2) address this question through a comparative evaluation of three strategies operating at different abstraction levels. Workflow decomposition reduces idle classical reservations by separating the quantum and classical phases of a hybrid job into independently scheduled steps. Dynamic malleability allows a running application to release classical nodes during long quantum phases and reacquire them when needed, improving cluster utilization without requiring an explicit workflow declaration. The vQPU approach, discussed under RQ3, provides a complementary transparent mechanism at the resource-management level. The performance results in Section 5.6 show that no single strategy universally dominates: workflow decomposition is most effective when phase structure can be declared in advance; malleability is most effective when phase durations are variable and hard to predict; and virtualization provides the greatest benefit when quantum phases are short and jobs can be safely interleaved.

8.2 Summary of Contributions

The following contributions, enumerated in Section 1.3, are here revisited with the results obtained.

1. **Systematic analysis of hybrid QML and combinatorial optimization paradigms.** Chapters 3 and 4 provide a structured examination of variational quantum models, Quantum Reservoir Computing, quantum kernel methods, QUBO-based formulations, and hybrid decomposition strategies, evaluated under realistic NISQ constraints. The case studies demonstrate that QML approaches avoiding variational training (QRC, QEK) are more robust to NISQ-level noise than their fully parameterized counterparts, and that hybrid decomposition (BBQ-mIS) can achieve optimal graph colorings in emulation while remaining experimentally viable on real hardware.

2. **Formal characterization of hybrid execution patterns.** The workload taxonomy and time-scale analysis of Chapter 5 establish a conceptual framework that connects algorithmic structure to infrastructure-level behavior. The analysis shows that latency, queueing, and synchronization overhead are not secondary engineering details but primary determinants of end-to-end performance for hybrid jobs in shared HPC environments.

3. **Design and implementation of the vQPU abstraction layer.** The vQPU architecture enables time-based multiplexing of quantum hardware within an HPC batch-scheduling framework. The prototype implementation is integrated with SLURM through a custom scheduling plugin and evaluated on representative hybrid workloads. Results confirm that the abstraction improves QPU utilization and reduces makespan for concurrent hybrid jobs while preserving transparency for user applications.

4. **Development and evaluation of platform-aware resource management strategies.** Three complementary strategies, loosely coupled workflows, dynamic malleability, and quantum resource virtualization, are analyzed and evaluated across different workload regimes and hardware targets. The comparative study of Section 5.7 provides quantitative guidance on strategy selection as a function of phase imbalance and predictability.

5. **Experimental validation on simulated and real infrastructures.** The experiments of Chapter 6 span noiseless statevector emulation, shot-based simulation, and real-device execution on QuEra's Aquila 256-qubit neutral-atom platform. The QRC pipeline achieves F1-scores on par with a DNN baseline in emulation on the full `CARDS_30000` dataset, with partial degradation on hardware that is absorbed by the classical readout at larger data scales. The QEK pipeline matches the Shortest Path Kernel on `PROTEINS256` (65.6% vs. 65.3% F1) and slightly exceeds it on the optimized `PROTEINS12` subset, including under hardware conditions. The BBQ-mIS workflow reproduces optimal graph colorings in emulation across all 120 tested instances; real-device execution on Aquila yields lower but non-trivial mIS recovery rates, validating the neutral-atom device as a viable offline candidate generator.

8.3 Limitations

The results of this dissertation were obtained under specific experimental conditions and their scope should be interpreted accordingly.

Scale of hardware experiments. All real-device executions were performed on instances whose size was constrained by QPU availability, Unit-Disk embedding requirements, and finite shot budgets. The QRC hardware campaign was limited to `CARDS_1000` and `CARDS_2500` subsets; the QEK hardware evaluation covered `PROTEINS12` and `PROTEINS256` after embedding-feasibility filtering. The conclusions regarding hardware performance apply within these bounds and may not extrapolate directly to larger or structurally different instances.

Scope of the BBQ-mIS hardware campaign. The real-hardware study for the optimization case study tested only three induced subgraphs extracted from a single 20-node graph-coloring instance derived from antenna positions in the city of Turin. Generalization of the hardware results to other graph families, problem sizes, or decomposition paths has not been experimentally validated.

Absence of closed-loop hardware execution. In all three case studies, the full hybrid workflow was not executed in a closed loop directly on the QPU. Remote and asynchronous cloud-like access to Aquila made tight classical–quantum synchronization impractical. The hardware studies therefore assess individual quantum phases or representative subproblems in isolation, rather than the complete end-to-end pipeline. This limitation reflects the operational constraints of early NISQ deployment rather than a methodological choice, but it means that latency and synchronization effects can only be characterized analytically and through emulation, not through direct measurement.

vQPU experiment on a single reserved device. The vQPU performance study was conducted on a real gate-based QPU (a 5-qubit IQM Spark device hosted at Politecnico di Torino) integrated with the Leonardo supercomputer at CINECA. However, the device was reserved exclusively for the experimental campaign to avoid external queueing variability. Even if machine load is simulated by varying the number of concurrent identical workloads, this does not reflect a genuine multi-tenant production scenario, in which backend-side calibration drift, vendor-imposed rate limits, and competing user jobs would introduce additional variability in queue times and circuit execution latency. Furthermore, the evaluation uses a single bench-

mark application (BBQ-mIS graph coloring via QAOA) and a single physical device; generalization to other QPU architectures or workload types has not been empirically validated. This is a common limitation in early-stage quantum systems research, where access to hardware, especially in the case of on-premise machine in HPC centers, is often limited and controlled, but it should be acknowledged when interpreting the results.

Malleability and workflow evaluation on emulated quantum workloads. Unlike the vQPU study, the performance characterization of the malleability and workflow strategies relied on an emulated quantum subroutine: a Simulated Annealing solver was used as a classical proxy for a quantum annealer, preserving the temporal structure of the hybrid workload without dispatching circuits to real QPU hardware. The scheduling and resource-utilization metrics obtained are therefore indicative of the strategies' benefits under realistic workload timing profiles, but do not capture backend-specific variability arising from real quantum execution.

Dataset subsets and embedding constraints. The QRC and QEK hardware datasets are strict subsets of the emulation datasets, both reduced to accommodate QPU register capacity and Unit-Disk embedding feasibility. Comparisons between hardware and emulation results are therefore made on matched subsets, and the performance gap should not be interpreted as representative of the algorithms' behavior at full dataset scale.

8.4 Future Work

The findings of this dissertation suggest several concrete directions for subsequent research.

Closed-loop hardware execution of hybrid iterative workflows. The most direct extension of the present work is to execute the QRC, QEK, and BBQ-mIS workflows in a tightly integrated online loop on real QPU hardware, as quantum access models mature toward lower-latency and more deterministic interfaces. This would allow direct measurement of the latency and synchronization overhead modeled analytically in this thesis, and would test whether the algorithmic and resource-management strategies developed here remain effective under closed-loop conditions.

Digital QRC on gate-based hardware. The QRC case study in this thesis relies on analog neutral-atom dynamics for reservoir evolution. A complementary line of investigation extends the reservoir computing paradigm to digital (gate-based) quantum circuits, where temporal data is encoded in rotation angles and partial measurement-and-reset operations provide the reservoir memory. Preliminary work applying this digital QRC framework to multi-step forecasting of ATM cash-demand timeseries on a real IQM Spark processor shows that, although the quantum models do not yet surpass classical baselines in absolute error metrics, they exhibit a partial ability to capture temporal structure as measured by Dynamic Time Warping. These results suggest that digital QRC may broaden the applicability of quantum reservoir approaches to gate-based near-term devices and to sequential prediction tasks beyond classification.

Scaling to larger problem instances. The case studies were necessarily constrained by NISQ-era hardware limitations. As QPU qubit counts and coherence times improve, extending the QRC reservoir to larger registers, the QEK to graph benchmarks beyond the UD-embedding constraints, and BBQ-mIS to graph instances with more than 20 nodes would provide a more comprehensive assessment of scalability. Particular attention should be paid to whether the cross-layer bottlenecks identified here shift or remain dominant at larger scales. In this direction, preliminary work on extending the graph coloring methodology has been undertaken: a hybrid quantum–classical Branch-and-Price formulation [2] reformulates the vertex coloring problem via column generation, delegating the pricing subproblem, a weighted Maximum Independent Set instance, to a quantum solver, thereby enabling decomposition strategies that scale beyond the bounds of the BBQ-mIS campaign reported in this thesis.

vQPU integration with heterogeneous quantum backends. The current vQPU prototype is designed and evaluated for a generic QPU interface. Extending the architecture to support multiple concurrent backends, superconducting, neutral-atom, and trapped-ion, within the same HPC environment would test the portability of the abstraction and introduce new challenges related to backend-specific compilation, calibration cycles, and cost models.

Adaptive and learning-based resource management. The resource management strategies evaluated in this thesis are essentially static in their policy logic: workflow phases are declared in advance, or malleability boundaries are coded at

design time. A natural extension is to incorporate online estimation of quantum phase durations and runtime adaptation of resource allocations, drawing on recent work on predictive HPC scheduling and reinforcement-learning-based job management.

Broader quantum algorithm families. This work focused on analog neutral-atom platforms for the QML and optimization case studies. Applying the same cross-layer methodology to gate-based superconducting or trapped-ion platforms would test the generality of the execution-model taxonomy and yield a richer comparative picture of how hardware modality shapes the algorithmic–infrastructure co-design space.

Error mitigation as a systems concern. The current experiments treat hardware noise primarily as an algorithmic challenge, with mitigation handled at the circuit or pulse level. A complementary approach is to integrate error-mitigation overhead, extra shots, post-processing complexity, calibration frequency, into the resource management layer, allowing the scheduler to make allocation decisions that account for the effective cost of operating near or below the noise threshold.

In summary, the NISQ era has established the practical disciplines of hybrid quantum computing: careful algorithmic scoping, layered execution modelling, and platform-aware resource management. The contributions of this dissertation demonstrate that these disciplines already produce measurable and reproducible results on real hardware. Future work will expand their reach as both hardware capabilities and integration infrastructures continue to mature.

References

[1] Giacomo Vitali, Chiara Vercellino, Paolo Viviani, Olivier Terzo, Bartolomeo Montrucchio, Valeria Zaffaroni, Francesca Cibrario, Christian Mattia, Giacomo Ranieri, Alessandro Sabatino, Francesco Bonazzi, and Davide Corbelletto. Quantum reservoir computing for credit card default prediction on a neutral atom platform. arXiv preprint arXiv:2510.04747 [quant-ph], 2025. Under review.

[2] Chiara Vercellino, M. Yassine Naghmouchi, Wesley Coelho, Giacomo Vitali, Alberto Scionti, Paolo Viviani, Olivier Terzo, and Bartolomeo Montrucchio. Hybrid quantum-classical branch-and-price method for the vertex coloring problem. arXiv preprint arXiv:2508.18887 [quant-ph], 2025. Under review.

[3] Marco Cipollini, Simone Rizzo, Sergio Iserte, Paolo Viviani, Giacomo Vitali, Matteo Barbieri, Gabriella Bettonte, Elisabetta Boella, Fulvio Ganz, Roberto Rocco, Orazio Spina, Antonio J. Peña, Petter Sandås, Iacopo Colonnelli, Alberto Scionti, Chiara Vercellino, Emanuele Dri, Jonathan Frassineti, Sara Marzella, Andrea Muratori, Daniele Ottaviani, Olivier Terzo, Bartolomeo Montrucchio, and Daniele Gregori. Three ways to share a qpu: Scheduling strategies for hybrid quantum-hpc applications. *Future Generation Computer Systems*, 185:108699, 2026. arXiv:2604.14955 [quant-ph].

[4] Paolo Viviani, Fabrizio Bertone, Giacomo Vitali, Emanuele Dri, Federico Stirano, Giuseppe Caragnano, Francesco Lubrano, Antonino Nespola, Olivier Terzo, Matteo Cocuzza, Bartolomeo Montrucchio, Giovanna Turvani, Gianluca Bertaina, Marco Coisson, Davide Calonico, Fabrizio Pirri, and Pietro Asinari. Lagrange: Operating italy's first publicly-accessible quantum computer for research and education. arXiv preprint arXiv:2604.21695 [quant-ph], 2026. Under review.

[5] Edoardo Giusto, Gabriele Iurlaro, Bartolomeo Montrucchio, Alberto Scionti, Olivier Terzo, Chiara Vercellino, Giacomo Vitali, and Paolo Viviani. Harnessing a 256-qubit neutral atom simulator for graph classification. In *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*, volume 01, pages 296–305, 2024.

[6] Chiara Vercellino, Giacomo Vitali, Paolo Viviani, Edoardo Giusto, Alberto Scionti, Andrea Scarabosio, Olivier Terzo, and Bartolomeo Montrucchio.

Bbq-mis: A parallel quantum algorithm for graph coloring problems. In *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, volume 02, pages 141–147, 2023.

[7] Paolo Viviani, Roberto Rocco, Matteo Barbieri, Gabriella Bettonte, Elisabetta Boella, Marco Cipollini, Jonathan Frassineti, Fulvio Ganz, Sara Marzella, Daniele Ottaviani, Simone Rizzo, Alberto Scionti, Chiara Vercellino, Giacomo Vitali, Olivier Terzo, Bartolomeo Montrucchio, and Daniele Gregori. Assessing the elephant in the room in scheduling for current hybrid hpc-qc clusters. In *2025 55th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pages 184–187, 2025.

[8] Roberto Rocco, Simone Rizzo, Matteo Barbieri, Gabriella Bettonte, Elisabetta Boella, Fulvio Ganz, Sergio Iserte, Antonio J. Peña, Petter Sandås, Alberto Scionti, Olivier Terzo, Chiara Vercellino, Giacomo Vitali, Paolo Viviani, Jonathan Frassineti, Sara Marzella, Daniele Ottaviani, Iacopo Colonnelli, and Daniele Gregori. Dynamic solutions for hybrid quantum-hpc resource allocation. In *2025 IEEE International Conference on Quantum Computing and Engineering (QCE)*, volume 02, pages 34–40, 2025.

[9] Chiara Vercellino, Paolo Viviani, Giacomo Vitali, Alberto Scionti, Andrea Scarabosio, Olivier Terzo, Edoardo Giusto, and Bartolomeo Montrucchio. Neural-powered unit disk graph embedding: qubits connectivity for some qubo problems. In *2022 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pages 186–196, 2022.

[10] Chiara Vercellino, Giacomo Vitali, Paolo Viviani, Alberto Scionti, Andrea Scarabosio, Olivier Terzo, Edoardo Giusto, and Bartolomeo Montrucchio. Neural optimization for quantum architectures: Graph embedding problems with distance encoder networks. In *2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC)*, pages 380–389, 2023.

[11] Chiara Vercellino, Giacomo Vitali, Paolo Viviani, Alberto Scionti, Olivier Terzo, and Bartolomeo Montrucchio. Harnessing den models for quantum computing tasks on neutral atom qpus. In *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*, volume 02, pages 217–222, 2024.

[12] Giuseppe Barillaro, Andrea Boella, Filippo Gandino, Mohammad Ghazi Vakili, Edoardo Giusto, Giovanni Mondo, Bartolomeo Montrucchio, Andrea Scarabosio, Alberto Scionti, Olivier Terzo, and Giacomo Vitali. Comparison of heuristic approaches to pci planning for quantum computers. In *2023 IEEE International Conference on Consumer Electronics (ICCE)*, pages 1–6, 2023.

[13] Paolo Savio, Alberto Scionti, Giacomo Vitali, Paolo Viviani, Chiara Vercellino, Olivier Terzo, Huy-Nam Nguyen, Donato Magarielli, Ennio Spano, Michele Marconcini, and Francesco Poli. Accelerating legacy applications with spatial computing devices. *The Journal of Supercomputing*, 79(7):7461–7483, 2023.

[14] Chiara Vercellino, Alberto Scionti, Giuseppe Varavallo, Paolo Viviani, Giacomo Vitali, and Olivier Terzo. A machine learning approach for an hpc use case: the jobs queuing time prediction. *Future Generation Computer Systems*, 143:215–230, 2023.

[15] Francesco Lubrano, Chiara Vercellino, Giacomo Vitali, Paolo Viviani, Alberto Scionti, and Olivier Terzo. Advanced resource allocation in the context of heterogeneous workflows management. In *Proceedings of the 2nd Workshop on Workflows in Distributed Environments*, WiDE '24, pages 14–20, New York, NY, USA, 2024. Association for Computing Machinery.

[16] Chiara Vercellino, Giacomo Vitali, Paolo Viviani, Alberto Scionti, Olivier Terzo, Bartolomeo Montrucchio, Pascal Jahan Elahi, and Ugo Varetto. Quantum-assisted design of space-terrestrial integrated networks. arXiv preprint arXiv:2602.04350 [quant-ph], 2026. Under review.

[17] John Preskill. Quantum computing in the NISQ era and beyond. *Quantum*, 2:79, 2018.

[18] Morten Kjaergaard, Mollie E. Schwartz, Jochen Braumüller, Philip Krantz, Jolin I-Jan Wang, Simon Gustavsson, and William D. Oliver. Superconducting qubits: Current state of play. *Annual Review of Condensed Matter Physics*, 11:369–395, 2020.

[19] Kishor Bharti, Alba Cervera-Lierta, Thi Ha Kyaw, Tobias Haug, Sumner Alperin-Lea, Abhinav Anand, Matthias Degroote, Hermanni Heimonen, Jakob S. Kottmann, Tim Menke, et al. Noisy intermediate-scale quantum algorithms. *Reviews of Modern Physics*, 94(1):015004, 2022.

[20] Christopher Monroe, Wesley C. Campbell, L-M Duan, Z-X Gong, Alexey V. Gorshkov, Paul W. Hess, Rajibul Islam, K. Kim, Norbert M. Linke, Guido Pagano, et al. Programmable quantum simulations of spin systems with trapped ions. *Reviews of Modern Physics*, 93(2):025001, 2021.

[21] Mark Saffman. Quantum computing with neutral atoms. *Progress in Quantum Electronics*, 73:100273, 2020.

[22] M. Saffman, T. G. Walker, and K. Mølmer. Quantum information with rydberg atoms. *Reviews of Modern Physics*, 82(3):2313–2363, August 2010.

[23] Antoine Browaeys and Thierry Lahaye. Many-body physics with individually controlled rydberg atoms. *Nature Physics*, 16(2):132–142, January 2020.

[24] Sergei Slussarenko and Geoff J. Pryde. Photonic quantum information processing: A review. *Applied Physics Reviews*, 6(4):041303, 2019.

[25] Timothy Proctor, Kenneth Rudinger, Kevin Young, Erik Nielsen, and Robin Blume-Kohout. Scalable benchmarking of quantum computers with random circuits. *Nature*, 601(7894):538–542, 2022.

[26] Paul D. Nation, Hwajung Kang, Neereja Sundaresan, and Jay M. Gambetta. Iterative quantum readout error mitigation. *PRX Quantum*, 2(4):040326, 2021.

[27] Jay M. Gambetta, A. D. Córcoles, S. T. Merkel, B. R. Johnson, John A. Smolin, Jerry M. Chow, Colm A. Ryan, Chad Rigetti, S. Poletto, Thomas A. Ohki, Mark B. Ketchen, and M. Steffen. Characterization of addressability by simultaneous randomized benchmarking. *Physical Review Letters*, 109(24), December 2012.

[28] Sergey Bravyi, Sheldon Kanazawa, Dmitri Maslov, Antonio Mezzacapo, and Jay M. Gambetta. Mitigating measurement errors in multiqubit experiments. *Physical Review A*, 103(4):042605, 2021.

[29] Zhenyu Cai, Ryan Babbush, Simon C. Benjamin, Suguru Endo, William J. Huggins, Ying Li, Jarrod R. McClean, and Bryan O'Gorman. Resource-efficient quantum error mitigation. *PRX Quantum*, 4(2):020101, 2023.

[30] Kristan Temme, Sergey Bravyi, and Jay M. Gambetta. Error mitigation for short-depth quantum circuits. *Physical Review Letters*, 119(18):180509, 2017.

[31] Suguru Endo, Simon C. Benjamin, and Ying Li. Practical quantum error mitigation for near-term devices. *Physical Review X*, 8(3):031027, 2018.

[32] Lorenza Viola, Emanuel Knill, and Seth Lloyd. Dynamical decoupling of open quantum systems. *Physical Review Letters*, 82(12):2417–2421, March 1999.

[33] Gushu Li, Yufei Ding, and Yuan Xie. Tackling the qubit mapping problem for nisq-era quantum devices. arXiv preprint arXiv:1809.02573 [cs.ET], 2019.

[34] Prakash Murali, Jonathan M. Baker, Ali Javadi-Abhari, Frederic T. Chong, and Margaret Martonosi. Noise-adaptive compiler mappings for noisy intermediate-scale quantum computers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 1015–1029, 2019.

[35] Zhenghao Zhou, Sanjiang Li, and Yuan Feng. Hardware-aware quantum circuit compilation via reinforcement learning. *npj Quantum Information*, 8(1):110, 2022.

[36] Emanuel Knill, Raymond Laflamme, and Wojciech H. Zurek. Resilient quantum computation: error models and thresholds. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 454(1969):365–384, January 1998.

[37] Google Quantum AI. Suppressing quantum errors by scaling a surface code logical qubit. *Nature*, 614(7949):676–681, 2023.

[38] Christopher Chamberland and Michael E. Beverland. Flag fault-tolerant error correction with arbitrary distance codes. *Quantum*, 2:53, February 2018.

[39] C. Ryan-Anderson, J. G. Bohnet, K. Lee, D. Gresh, A. Hankin, J. P. Gaebler, D. Francois, A. Chernoguzov, D. Lucchetti, N. C. Brown, T. M. Gatterman, S. K. Halit, K. Gilmore, J. A. Gerber, B. Neyenhuis, D. Hayes, and R. P. Stutz. Realization of real-time fault-tolerant quantum error correction. *Phys. Rev. X*, 11:041058, Dec 2021.

[40] Craig Gidney and Martin Ekerå. How to factor 2048-bit RSA integers in 8 hours using 20 million noisy qubits. *Quantum*, 5:433, 2021.

[41] Marco Cerezo, Andrew Arrasmith, Ryan Babbush, Simon C. Benjamin, Suguru Endo, Mikhail G. Lukin, Jarrod R. McClean, Kosuke Mitarai, Xiao Yuan, Lukasz Bozhedarov, et al. Variational quantum algorithms. *Nature Reviews Physics*, 3(9):625–644, 2021.

[42] Sukin Sim, Jonathan Romero, Peter D. Jerome, Jarrod R. McClean, Hsin-Yuan Huang, Jakob S. Kottmann, Murphy Wang, Kevin Radin, and Alán Aspuru-Guzik. Expressibility and trainability of parameterized quantum circuits for quantum machine learning. *PRX Quantum*, 3(1):010313, 2022.

[43] Andrea Skolik, Jarrod R. McClean, Masoud Mohseni, Patrick van der Smagt, and Martin Leib. Layerwise learning for quantum neural networks. *Quantum Machine Intelligence*, 3(1), January 2021.

[44] Harper R. Grimsley, Sophia E. Economou, Edwin Barnes, and Nicholas J. Mayhall. An adaptive variational algorithm for exact molecular simulations on a quantum computer. *Nature Communications*, 10(1), July 2019.

[45] Arthur Pesah, M. Cerezo, Samson Wang, Tyler Volkoff, Andrew T. Sornborger, and Patrick J. Coles. Absence of barren plateaus in quantum convolutional neural networks. *Physical Review X*, 11(4), October 2021.

[46] Jarrod R. McClean, Sergio Boixo, Vadim N. Smelyanskiy, Ryan Babbush, and Hartmut Neven. Barren plateaus in quantum neural network training landscapes. *Nature Communications*, 9(1):4812, 2018.

[47] Marco Cerezo, Akira Sone, Tyler Volkoff, Lukasz Cincio, and Patrick J. Coles. Cost function dependent barren plateaus in shallow parameterized quantum circuits. *Nature Communications*, 12(1):1791, 2021.

[48] Roeland Wiersema, Cunlu Zhou, Yisun de Sereville, Juan Felipe Carrasquilla, Isaac B. Kim, and Hsin-Yuan Huang. Classification of gradient variance scaling in VQA landscapes. *Physical Review Research*, 5(1):013146, 2023.

[49] Youle Li, Yilun Huang, Siyuan Wang, Sanjiang Li, and Yuan Feng. Distributed variational quantum algorithms. *Quantum Science and Technology*, 8(4):045014, 2023.

[50] Jacob Biamonte, Peter Wittek, Nicola Nicola, Patrick Rebentrost, Nathan Wiebe, and Seth Lloyd. Quantum machine learning. *Nature*, 549(7671):195–202, 2017.

[51] Hsin-Yuan Huang, Michael Broughton, Masoud Mohseni, Ryan Babbush, Sergio Boixo, Hartmut Neven, and Jarrod R. McClean. Power of data in quantum machine learning. *Nature Communications*, 12(1), May 2021.

[52] Giuseppe Carleo, Yusuke Nomura, and Masatoshi Imada. Perspectives on quantum machine learning. *Nature Reviews Physics*, 4:12–13, 2022.

[53] Maria Schuld and Nathan Killoran. Quantum machine learning in feature hilbert spaces. *Phys. Rev. Lett.*, 122:040504, Feb 2019.

[54] Vojtěch Havlíček, Antonio D. Córcoles, Kristan Temme, Aram W. Harrow, Abhinav Kandala, Jerry M. Chow, and Jay M. Gambetta. Supervised learning with quantum-enhanced feature spaces. *Nature*, 567(7747):209–212, 2019.

[55] Samson Wang, Enrico Fontana, M. Cerezo, Kunal Sharma, Akira Sone, Lukasz Cincio, and Patrick J. Coles. Noise-induced barren plateaus in variational quantum algorithms. *Nature Communications*, 12(1), November 2021.

[56] Amira Abbas, David Sutter, Christa Zoufal, Aurelien Lucchi, Alessio Figalli, and Stefan Woerner. The power of quantum neural networks. *Nature Computational Science*, 1(6):403–409, 2021.

[57] Michael Ragone, Peter Bakalov, Jonathan E. Brande, Jacob Bringewatt, Akira Sone, Andrew J. Berkley, Paul D. Bunyk, Trevor Lanting, Mark W. Johnson, and Mohammad H. Amin. Noise-aware training for quantum machine learning. *Quantum Machine Intelligence*, 4(2):20, 2022.

[58] Ewin Tang. A quantum-inspired classical algorithm for recommendation systems. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2019, page 217–228, New York, NY, USA, 2019. Association for Computing Machinery.

[59] Ewin Tang. Quantum principal component analysis only achieves an exponential speedup because of its state preparation assumptions. *Physical Review Letters*, 127(6), August 2021.

[60] Supanut Thanasilp, Samson Wang, M. Cerezo, and Zoë Holmes. Exponential concentration in quantum kernel methods. arXiv preprint arXiv:2208.11060 [quant-ph], 2024.

[61] Keisuke Fujii and Kohei Nakajima. Harnessing quantum dynamics as a machine learning resource. *Physical Review Applied*, 8(2):024030, 2017.

[62] A. Kutvonen, K. Fujii, and K. Nakajima. Experimental quantum reservoir computing with a superconducting processor. *Scientific Reports*, 12(1):10134, 2022.

[63] Andrew Lucas. Ising formulations of many NP problems. *Frontiers in Physics*, 2:5, 2014.

[64] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm. *arXiv preprint arXiv:1411.4028*, 2014.

[65] V. Akshay, H. Philms, M. E. S. Morales, and Jacob Biamonte. Reachability deficits in quantum approximate optimization. *PRX Quantum*, 2(2):020346, 2021.

[66] Jonathan Wurtz and Peter J. Love. Counterdiabaticity and the quantum approximate optimization algorithm. *Quantum*, 6:635, January 2022.

[67] Leo Zhou, Sheng-Tao Wang, Soonwon Choi, Hannes Pichler, and Mikhail G. Lukin. Quantum approximate optimization algorithm: Performance, gauge invariance, and symmetry protection. *Physical Review X*, 10(2):021067, 2020.

[68] Daniel J. Egger, Claudio Gambella, Jakub Marecek, Scott McFaddin, Martin Michiels, Mario Ritkay, Stefan Zugno, and Stefan Woerner. Warm-starting quantum optimization with classical mixed-integer programming. *Physical Review Applied*, 15(2):024060, 2021.

[69] Ruslan Shaydulin and Marco Pistoia. A hybrid approach for readout error mitigation in QAOA. *Quantum Science and Technology*, 8(3):035005, 2023.

[70] Elijah Pelofske, Georg Hahn, and Hristo Djidjev. Short-depth QAOA for the maximum cut problem on large-scale graphs. *Scientific Reports*, 12(1):20038, 2022.

[71] Prashant Date, Davis Arthur, and Lauren Pusey-Nazzaro. QUBO decomposition strategies for the vehicle routing problem. *Quantum Information Processing*, 20:1–23, 2021.

[72] Tobias Stollenwerk, Elisabeth Lobe, Marc Jung, et al. Large neighborhood search using a quantum annealer for microchip layout planning. *Quantum*, 6:758, 2022.

[73] Lennart Bittel and Martin Kliesch. Training variational quantum algorithms is NP-hard. *Physical Review Letters*, 127(12):120502, 2021.

[74] Andy B. Yoo, Morris A. Jette, and Mark Grondona. SLURM: Simple linux utility for resource management. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 44–60. Springer, 2003.

[75] Dror G. Feitelson. *Workload Modeling for Computer Systems Performance Evaluation*. Cambridge University Press, 2020.

[76] Alexander Leonenkov et al. Malleable job scheduling in HPC systems. *Journal of Parallel and Distributed Computing*, 161:1–15, 2022.

[77] Jack Dongarra et al. The international exascale software project roadmap. *International Journal of High Performance Computing Applications*, 36, 2022.

[78] Marco A. S. Netto, Rodrigo N. Calheiros, Eduardo R. Rodrigues, Renato L. F. Cunha, and Rajkumar Buyya. Hpc cloud for scientific and business applications: Taxonomy, vision, and research challenges. *ACM Comput. Surv.*, 51(1), January 2018.

[79] Ali Javadi-Abhari, Matthew Treinish, Kevin Krsulich, Christopher J. Wood, Jake Lishman, Julien Gacon, Simon Martiel, Paul D. Nation, Lev S. Bishop, Andrew W. Cross, Blake R. Johnson, and Jay M. Gambetta. Quantum computing with Qiskit. arXiv preprint arXiv:2405.08810 [quant-ph], 2024.

[80] Rodney Van Meter. *Quantum Computer Architecture*. Morgan & Claypool, 2022.

[81] Marco Pistoia et al. The dawn of quantum computing in enterprise applications. *IEEE Computer*, 54(10):14–23, 2021.

[82] Travis S. Humble et al. HPC–QC integration: Challenges and opportunities. *IEEE Micro*, 43(1):10–18, 2023.

[83] Judit Giménez et al. Co-scheduling accelerators in HPC clusters. *Future Generation Computer Systems*, 115, 2021.

[84] Wiesław Lubiński et al. Optimization of quantum computer workloads: A survey. *ACM Computing Surveys*, 2023.

[85] Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum algorithm for linear systems of equations. *Physical Review Letters*, 103(15):150502, 2009.

[86] Marcello Benedetti, Erika Lloyd, Stefan Sack, and Mattia Fiorentini. Parameterized quantum circuits as machine learning models. *Quantum Science and Technology*, 4(4):043001, 2019.

[87] Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. The quest for a quantum neural network. *Quantum Information Processing*, 13:2567–2586, 2014.

[88] Adrián Pérez-Salinas, Alba Cervera-Lierta, Elies Gil-Fuster, and José I. Latorre. Data re-uploading for a universal quantum classifier. *Quantum*, 4:226, 2020.

[89] Herbert Jaeger. The "echo state" approach to analysing and training recurrent neural networks. *GMD Report*, 2001.

[90] Wolfgang Maass, Thomas Natschläger, and Henry Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, 14(11):2531–2560, 2002.

[91] Keisuke Fujii and Kohei Nakajima. Harnessing disordered-ensemble quantum dynamics for machine learning. *Physical Review Applied*, 8:024030, 2017.

[92] Milan Kornjača, Hong-Ye Hu, Chen Zhao, Jonathan Wurtz, Phillip Weinberg, Majd Hamdan, Andrii Zhdanov, Sergio H. Cantu, Hengyun Zhou, Rodrigo Araiza Bravo, Kevin Bagnall, James I. Basham, Joseph Campo, Adam Choukri, Robert DeAngelo, Paige Frederick, David Haines, Julian Hammett, Ning Hsu, Ming-Guang Hu, Florian Huber, Paul Niklas Jepsen, Ningyuan Jia, Thomas Karolyshyn, Minho Kwon, John Long, Jonathan Lopatin, Alexander Lukin, Tommaso Macrì, Ognjen Marković, Luis A. Martínez-Martínez, Xianmei Meng, Evgeny Ostroumov, David Paquette, John Robinson, Pedro Sales Rodriguez, Anshuman Singh, Nandan Sinha, Henry Thoreen, Noel Wan, Daniel Waxman-Lenz, Tak Wong, Kai-Hsin Wu, Pedro L. S. Lopes, Yuval Boger, Nathan Gemelke, Takuya Kitagawa, Alexander Keesling, Xun Gao, Alexei Bylinskii, Susanne F. Yelin, Fangli Liu, and Sheng-Tao Wang. Large-scale quantum reservoir learning with an analog quantum computer. arXiv preprint arXiv:2407.02553 [quant-ph], 2024.

[93] I-Cheng Yeh. Default of Credit Card Clients. UCI Machine Learning Repository, 2009. DOI: https://doi.org/10.24432/C55S3H.

[94] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *J. Artif. Int. Res.*, 16(1):321–357, June 2002.

[95] Wei-Chao Lin, Chih-Fong Tsai, Ya-Han Hu, and Jing-Shang Jhang. Clustering-based undersampling in class-imbalanced data. *Information Sciences*, 409-410:17–26, 2017.

[96] Toshiki Yasuda, Yudai Suzuki, Tomoyuki Kubota, Kohei Nakajima, Qi Gao, Wenlong Zhang, Satoshi Shimono, Hendra I. Nurdin, and Naoki Yamamoto. Quantum reservoir computing with repeated measurements on superconducting devices. *arXiv preprint arXiv:2310.06706 [quant-ph]*, 10 2023.

[97] Jonathan Wurtz et al. Aquila: QuEra's 256-qubit neutral-atom quantum computer. *arXiv preprint arXiv:2306.11727 [quant-ph]*, 6 2023.

[98] Louis-Paul Henry, Slimane Thabet, Constantin Dalyac, and Loïc Henriet. Quantum evolution kernel: Machine learning on graphs with programmable arrays of qubits. *Phys. Rev. A*, 104:032416, Sep 2021.

[99] Peter I. Frazier. A tutorial on bayesian optimization. arXiv preprint arXiv:1807.02811 [stat.ML], 2018.

[100] Paul D Dobson and Andrew J Doig. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of molecular biology*, 330(4):771–783, 2003.

[101] Karsten M Borgwardt, Cheng Soon Ong, Stefan Schönauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(suppl_1):i47–i56, 2005.

[102] Bloqade.jl: Package for the quantum computation and quantum simulation based on the neutral-atom architecture. Software package, QuEra Computing, https://github.com/QuEraComputing/Bloqade.jl/, 2023.

[103] K.M. Borgwardt and H.P. Kriegel. Shortest-path kernels on graphs. In *Fifth IEEE International Conference on Data Mining (ICDM'05)*, pages 8 pp.–, 2005.

[104] Alexander Cowtan, Silas Dilkes, Ross Duncan, Alexandre Krajenbrink, Will Simmons, and Seyon Sivarajah. On the Qubit Routing Problem. *Leibniz Int. Proc. Inf.*, 135:5:1–5:32, 2019.

[105] Jarrod R. McClean, Jonathan Romero, Ryan Babbush, and Alán Aspuru-Guzik. The theory of variational hybrid quantum-classical algorithms. *New Journal of Physics*, 18(2):023023, 2016.

[106] Daniel J. Egger, Max Werninghaus, Marc Ganzhorn, Gian Salis, Andreas Fuhrer, Philipp Mueller, and Woerner Stefan. One-pulse test of quantum-classical optimization. *Physical Review Applied*, 15(4):044007, 2021.

[107] Abhinav Kandala, Kristan Temme, Antonio D Córcoles, Antonio Mezzacapo, Jerry M Chow, and Jay M Gambetta. Error mitigation extends the computational reach of a noisy quantum processor. *Nature*, 567(7749):491—495, March 2019.

[108] Robert Endre Tarjan and Anthony E. Trojanowski. Finding a maximum independent set. *SIAM Journal on Computing*, 6(3):537–546, 1977.

[109] Tommy R Jensen and Bjarne Toft. *Graph coloring problems*. John Wiley & Sons, 2011.

[110] Susan Khor. Application of graph colouring to biological networks. *IET systems biology*, 4(3):185–192, 2010.

[111] R. Acedo-Hernández, M. Toril, S. Luna-Ramírez, and C. Úbeda. A pci planning algorithm for jointly reducing reference signal collisions in lte uplink and downlink. *Computer Networks*, 119:112–123, 2017.

[112] Gregory J Chaitin, Marc A Auslander, Ashok K Chandra, John Cocke, Martin E Hopkins, and Peter W Markstein. Register allocation via coloring. *Computer languages*, 6(1):47–57, 1981.

[113] Tadashi Kadowaki and Hidetoshi Nishimori. Quantum annealing in the transverse ising model. *Physical Review E*, 58(5):5355–5363, November 1998.

[114] Stuart Hadfield, Zhihui Wang, Bryan O'Gorman, Eleanor G. Rieffel, Davide Venturelli, and Rupak Biswas. From the quantum approximate optimization algorithm to a quantum alternating operator ansatz. *Algorithms*, 12(2):34, February 2019.

[115] Andrew Lucas. Ising formulations of many NP problems. *Frontiers in Physics*, 2:5, 2014.

[116] Tameem Albash and Daniel A. Lidar. Adiabatic quantum computation. *Reviews of Modern Physics*, 90(1), January 2018.

[117] Tyler Christeson, Md Habib Ullah, Ali Arabnya, Amin Khodaei, and Rui Fan. Hybrid quantum-classical optimization of the resource scheduling problem. arXiv preprint arXiv:2511.00733 [eess.SY], 2025.

[118] Zedong Peng, Daniel de Roux, and David E. Bernal Neira. Hybrid quantum branch-and-bound method for quadratic unconstrained binary optimization. arXiv preprint arXiv:2509.11040 [math.OC], 2025.

[119] Sandeep Suresh Cranganore, Vincenzo De Maio, Ivona Brandic, and Ewa Deelman. Paving the way to hybrid quantum–classical scientific workflows. *Future Generation Computer Systems*, 158:346–366, 2024.

[120] Pradeep Mantha, Florian J. Kiwit, Nishant Saurabh, Shantenu Jha, and Andre Luckow. Pilot-quantum: A quantum-hpc middleware for resource, workload and task management. arXiv preprint arXiv:2412.18519 [quant-ph], 2025.

[121] Amr Elsharkawy, Xiaorang Guo, and Martin Schulz. Integration of Quantum Accelerators into HPC: Toward a Unified Quantum Platform. In *2024 International Conference on Quantum Computing and Engineering*, 7 2024.

[122] Thomas Beck, Alessandro Baroni, Ryan Bennink, Gilles Buchs, Eduardo Antonio Coello Pérez, Markus Eisenbach, Rafael Ferreira da Silva, Muralikrishnan Gopalakrishnan Meena, Kalyan Gottiparthi, Peter Groszkowski, Travis S. Humble, Ryan Landfield, Ketan Maheshwari, Sarp Oral, Michael A. Sandoval, Amir Shehata, In-Saeng Suh, and Christopher Zimmer. Integrating quantum computing resources into scientific hpc ecosystems. *Future Generation Computer Systems*, 161:11–25, December 2024.

[123] Amir Shehata, Peter Groszkowski, Thomas Naughton, Murali Gopalakrishnan Meena, Elaine Wong, Daniel Claudino, Rafael Ferreira da Silvaa, and Thomas Beck. Bridging paradigms: Designing for hpc-quantum convergence. arXiv preprint arXiv:2503.01787 [quant-ph], 2025.

[124] Tianyi Peng, Aram W Harrow, Maris Ozols, and Xiaodi Wu. Simulating large quantum circuits on a small quantum computer. *Physical review letters*, 125(15):150504, 2020.

[125] Rafael Ferreira da Silva, Rosa M. Badia, Venkat Bala, Debbie Bard, Peer-Timo Bremer, Ian Buckley, Silvina Caino-Lores, Kyle Chard, Carole Goble, Shantenu Jha, Daniel S. Katz, Daniel Laney, Manish Parashar, Frederic Suter, Nick Tyler, Thomas Uram, Ilkay Altintas, Stefan Andersson, William Arndt, Juan Aznar, Jonathan Bader, Bartosz Balis, Chris Blanton, Kelly Rosa Braghetto, Aharon Brodutch, Paul Brunk, Henri Casanova, Alba Cervera

Lierta, Justin Chigu, Taina Coleman, Nick Collier, Iacopo Colonnelli, Frederik Coppens, Michael Crusoe, Will Cunningham, Bruno de Paula Kinoshita, Paolo Di Tommaso, Charles Doutriaux, Matthew Downton, Wael Elwasif, Bjoern Enders, Chris Erdmann, Thomas Fahringer, Ludmilla Figueiredo, Rosa Filgueira, Martin Foltin, Anne Fouilloux, Luiz Gadelha, Andy Gallo, Artur Garcia Saez, Daniel Garijo, Roman Gerlach, Ryan Grant, Samuel Grayson, Patricia Grubel, Johan Gustafsson, Valerie Hayot-Sasson, Oscar Hernandez, Marcus Hilbrich, AnnMary Justine, Ian Laflotte, Fabian Lehmann, Andre Luckow, Jakob Luettgau, Ketan Maheshwari, Motohiko Matsuda, Doriana Medic, Pete Mendygral, Marek Michalewicz, Jorji Nonaka, Maciej Pawlik, Loic Pottier, Line Pouchard, Mathias Putz, Santosh Kumar Radha, Lavanya Ramakrishnan, Sashko Ristov, Paul Romano, Daniel Rosendo, Martin Ruefenacht, Katarzyna Rycerz, Nishant Saurabh, Volodymyr Savchenko, Martin Schulz, Christine Simpson, Raul Sirvent, Tyler Skluzacek, Stian Soiland-Reyes, Renan Souza, Sreenivas Rangan Sukumar, Ziheng Sun, Alan Sussman, Douglas Thain, Mikhail Titov, Benjamin Tovar, Aalap Tripathy, Matteo Turilli, Bartosz Tuznik, Hubertus van Dam, Aurelio Vivas, Logan Ward, Patrick Widener, Sean Wilkinson, Justyna Zawalska, and Mahnoor Zulfiqar. Workflows community summit 2022: A roadmap revolution. Technical report, Zenodo, doi:10.5281/zenodo.7750670, March 2023.

[126] Frédéric Suter, Tainã Coleman, undefinedlkay Altintaş, Rosa M. Badia, Bartosz Balis, Kyle Chard, Iacopo Colonnelli, Ewa Deelman, Paolo Di Tommaso, Thomas Fahringer, Carole Goble, Shantenu Jha, Daniel S. Katz, Johannes Köster, Ulf Leser, Kshitij Mehta, Hilary Oliver, J.-Luc Peterson, Giovanni Pizzi, Loïc Pottier, Raül Sirvent, Eric Suchyta, Douglas Thain, Sean R. Wilkinson, Justin M. Wozniak, and Rafael Ferreira da Silva. A terminology for scientific workflow systems. *Future Gener. Comput. Syst.*, 174(C), January 2026.

[127] Vincenzo De Maio, Dominik Bork, and Ivona Brandic. Rigoletto: A workflow definition language for hybrid quantum-classical scientific applications. In *2024 26th International Conference on Business Informatics (CBI)*, pages 40–49, 2024.

[128] Iacopo Colonnelli, Barbara Cantalupo, Ivan Merelli, and Marco Aldinucci. Streamflow: Cross-breeding cloud with hpc. *IEEE Transactions on Emerging Topics in Computing*, 9(4):1723–1737, 2021.

[129] Michael R. Crusoe, Sanne Abeln, Alexandru Iosup, Peter Amstutz, John Chilton, Nebojša Tijanić, Hervé Ménager, Stian Soiland-Reyes, Bogdan Gavrilović, Carole Goble, and The CWL Community. Methods included: standardizing computational reuse and portability with the common workflow language. *Commun. ACM*, 65(6):54–63, May 2022.

[130] Sergio Iserte, Rafael Mayo, Enrique S. Quintana-Ortí, and Antonio J. Peña. Dmrlib: Easy-coding and efficient resource management for job malleability. *IEEE Transactions on Computers*, 70(9):1443–1457, 2021.

[131] Nan Li and Longin Jan Latecki. Clustering aggregation as maximum-weight independent set. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, page 782–790, Red Hook, NY, USA, 2012. Curran Associates Inc.

[132] J. MacQueen. Some methods for classification and analysis of multivariate observations. In L. M. Le Cam and J. Neyman, editors, *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability - Vol. 1*, pages 281–297. University of California Press, Berkeley, CA, USA, 1967.

[133] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, page 226–231. AAAI Press, 1996.

[134] Daniel Müllner. fastcluster: Fast hierarchical, agglomerative clustering routines for r and python. *Journal of Statistical Software*, 53(9):1–18, 2013.

[135] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[136] Riccardo Scotti, Gabriella Bettonte, Antonio Costantini, Sara Marzella, Daniele Ottaviani, and Stefano Lodi. A clustering aggregation algorithm on neutral-atoms and annealing quantum processors. arXiv preprint arXiv:2412.07558 [quant-ph], 2024.

[137] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning: with Applications in R*. Springer Publishing Company, Incorporated, 2014.

[138] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in python. *J. Mach. Learn. Res.*, 12(null):2825–2830, November 2011.

[139] Sagar Imambi, Kolla Bhanu Prakash, and GR Kanagachidambaresan. Pytorch. *Programming with TensorFlow: solution for edge computing applications*, pages 87–104, 2021.

[140] Brent N. Clark, Charles J. Colbourn, and David S. Johnson. Unit disk graphs. *Discrete Math.*, 86(1–3):165–177, January 1991.

[141] Coen Bron and Joep Kerbosch. Algorithm 457: finding all cliques of an undirected graph. *Commun. ACM*, 16(9):575–577, September 1973.

[142] Etsuji Tomita, Akira Tanaka, and Haruhisa Takahashi. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theor. Comput. Sci.*, 363(1):28–42, October 2006.

[143] Eugene L. Lawler. A note on the complexity of the chromatic number problem. *Inf. Process. Lett.*, 5:66–67, 1976.

[144] Henrique Silvério, Sebastián Grijalva, Constantin Dalyac, Lucas Leclerc, Peter J. Karalekas, Nathan Shammah, Mourad Beji, Louis-Paul Henry, and Loïc Henriet. Pulser: An open-source package for the design of pulse sequences in programmable neutral-atom arrays. *Quantum*, 6:629, January 2022.

[145] Bob Bixby. The gurobi optimizer. *Transp. Re-search Part B*, 41(2):159–178, 2007.

[146] A. J. Hoffman and Leonard Howes. On eigenvalues and colorings of graphs, ii. *Annals of the New York Academy of Sciences*, 175(1):238–242, 1970.

[147] Clive Elphick and Pawel Wocjan. An inertial lower bound for the chromatic number of a graph. *The Electronic Journal of Combinatorics*, 24(1):P1.58, Mar. 2017.

[148] C.S. Edwards and C.H. Elphick. Lower bounds for the clique and the chromatic numbers of a graph. *Discrete Applied Mathematics*, 5(1):51–64, 1983.

[149] L. Lovasz. On the shannon capacity of a graph. *IEEE Transactions on Information Theory*, 25(1):1–7, 1979.

[150] D. J. A. Welsh and M. B. Powell. An upper bound for the chromatic number of a graph and its application to timetabling problems. *The Computer Journal*, 10(1):85–86, 01 1967.

[151] Thomas M. J. Fruchterman and Edward M. Reingold. Graph drawing by force-directed placement. *Software: Practice and Experience*, 21(11):1129–1164, 1991.

[152] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019.

Appendix A

Details of Implemented Software Components

A.1 DEN Model for Unit-Disk Graph Embedding

This section describes the Distance Encoder Network (DEN), the neural-based solver used to embed problem graphs as unit-disk graphs (UDGs) on the neutral-atom quantum registers employed in this work. The embedding step is a prerequisite for executing both the QEK and BBQ-mIS algorithms on Aquila and Orion Alpha hardware. Full technical details are presented in [9–11].

A.1.1 The Constrained Unit Disk Graph Problem

In a neutral-atom register, each qubit corresponds to a Rydberg atom placed at a specific (x, y) position. Two atoms interact via the Rydberg blockade effect when their distance falls below the blockade radius r_b, which depends on the laser Rabi frequency Ω and the C_6 van der Waals coefficient. This interaction induces a unit-disk graph (UDG): an undirected graph $\mathscr{G}(\mathscr{V}, \mathscr{E})$ in which an edge (i, j) exists if and only if $\|p_i - p_j\|_2 \leq r_b$.

Encoding a target problem graph onto the hardware therefore requires finding atom positions that reproduce the desired adjacency pattern. Hardware constraints bound the feasible region: positions must lie within the register area, atoms must not

be placed closer than a minimum separation D_{min}, and the blockade radius sets the maximum adjacency distance $D_{adj} \approx r_b$.

Formally, given graph $\mathscr{G}(\mathscr{V}, \mathscr{E})$ with $n = |\mathscr{V}|$ vertices, the *Constrained Unit Disk Graph (CUDG)* problem seeks atom coordinates $\overrightarrow{p}_i \in \mathbb{R}^2$, $\forall i \in \mathscr{V}$, such that:

$$\|\overrightarrow{p}_i - \overrightarrow{p}_j\|_2 \in [D_{min}, D_{adj}] \qquad \forall (i,j) \in \mathscr{E} \tag{A.1}$$

$$\|\overrightarrow{p}_i - \overrightarrow{p}_j\|_2 > D_{adj} \qquad \forall (i,j) \notin \mathscr{E} \tag{A.2}$$

$$\overrightarrow{p}_i \in \mathscr{D}_{\text{reg}} \qquad \forall i \in \mathscr{V} \tag{A.3}$$

where $\mathscr{D}_{\text{reg}}$ is the register domain. Beyond feasibility, a good embedding maximises the *adjacency gap* $\alpha = \min_{(i,j)\notin\mathscr{E}} \|\overrightarrow{p}_i - \overrightarrow{p}_j\|_2 - D_{adj}$, which quantifies the separation between adjacent and non-adjacent distances and improves robustness to positional noise.

The CUDG problem is NP-hard in general. Classical non-linear solvers (e.g., Ipopt) achieve near-zero success rates even for $n \geq 20$ under moderate time budgets [10]. Two structural properties, derived from Thue's hexagonal circle-packing theorem, bound the class of embeddable graphs under the hardware parameters $D_{min} = 4\,\mu$m and $D_{adj} \approx 10.26\,\mu$m [9]:

- **Maximum clique:** the largest fully-connected subgraph has at most 7 vertices ($|K_{\max}| \leq 7$).
- **Maximum degree:** no vertex may have more than 18 neighbours ($\Delta_{\max} \leq 18$).

Graphs violating these conditions have no feasible 2D embedding under the given hardware constraints.

A.1.2 DEN Architecture and Training Procedure

The Distance Encoder Network (DEN) is a neural-enhanced solver introduced in [10] to address the CUDG problem. Unlike a standard predictive model, the DEN acts as an optimisation solver: a separate model instance is trained for each graph, and training constitutes the solving process.

Overview

The overall framework, illustrated in Figure A.1, consists of two phases.

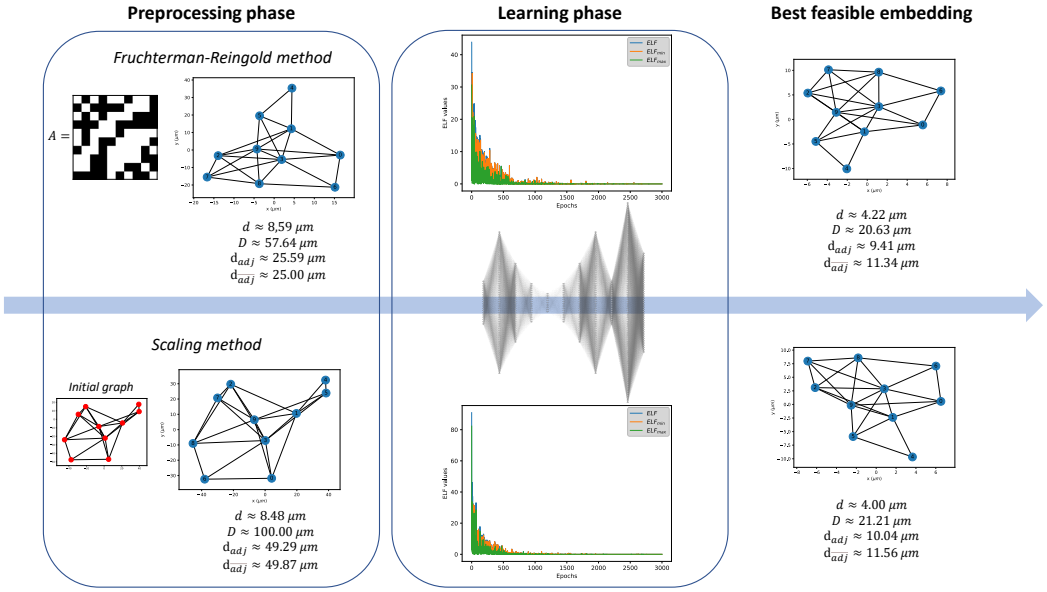


Fig. A.1 DEN optimisation pipeline. The preprocessing phase initialises vertex coordinates using either the Scaling or Fruchterman-Reingold method. The learning phase trains the DEN to find a feasible UDG embedding while maximising the adjacency gap. The best feasible embedding found across all training epochs is returned.

Preprocessing phase. Initial vertex coordinates $\overrightarrow{p}_i$ are set using one of two methods:

- *Scaling*: if the graph has associated geometric coordinates (e.g., antenna positions), these are rescaled to fit the register domain. This guarantees satisfaction of the upper-bound constraint (A.2) at initialisation.
- *Fruchterman-Reingold*: a force-directed layout algorithm [151] that models attractive forces between adjacent pairs and repulsive forces between all pairs. This is used when no initial coordinates are available, as is typical for general graphs in combinatorial optimisation tasks.

Neither method yields a CUDG-feasible solution; they provide warm-start coordinates that support faster convergence.

Learning phase. The DEN is trained for up to $E = 3000$ epochs on the single graph instance. At each epoch, a training step (with dropout) is followed by an inference step (without dropout); the latter computes the current embedding and checks feasibility. When a feasible solution is found, if the adjacency gap exceeds the best recorded value, the embedding is saved as the new incumbent. The parameter α in the loss function is updated to the best gap found so far, incrementally raising the optimisation target.

Network Architecture

The DEN has two components, described in Table A.1.

Table A.1 DEN architecture for a graph with n vertices embedded in $N = 2$ dimensions. All fully connected layers include bias nodes and dropout (probability p_{drop}), except the fixed-weight layers.

Component	Layer	Input size	Output size	Activation
Trainable autoencoder	FC (encoder)	$n \times N$	64	ReLU
	FC (encoder)	64	36	ReLU
	FC (encoder)	36	18	ReLU
	FC (encoder)	18	9	ReLU
	FC (decoder)	9	18	ReLU
	FC (decoder)	18	36	ReLU
	FC (decoder)	36	64	ReLU
	FC (*CoordL*)	64	$n \times N$	$L \cdot \tanh$
Fixed-weight distance calculator	*DiffL* (fixed ± 1)	$n \times N$	$N\binom{n}{2}$	$(\cdot)^2$
	SumL (fixed $+1$)	$N\binom{n}{2}$	$\binom{n}{2}$	identity

The *trainable autoencoder* maps input coordinates to new feasible coordinates. The output layer (*CoordL*) applies an $L \cdot \tanh$ activation that constrains coordinates to $(-L, +L)^2$, matching the register domain. The bottleneck (9 hidden units) provides a compact latent representation of the spatial layout.

The *fixed-weight distance calculator* makes the model geometrically aware. The difference layer (*DiffL*) computes all pairwise coordinate differences $\overrightarrow{p}_i^f - \overrightarrow{p}_j^f$ along each axis; a square activation then yields squared differences. The sum layer (*SumL*) aggregates per-axis squared differences to produce squared Euclidean distances $d_{ij}^2 = \|\overrightarrow{p}_i^f - \overrightarrow{p}_j^f\|_2^2$ for all $\binom{n}{2}$ pairs. These distances form the input to the loss function.

Embedding Loss Function (ELF)

The ELF translates the CUDG feasibility constraints into a differentiable penalty. It uses a margin ranking loss with target distances derived from the adjacency matrix A:

$$\mathrm{ELF}(v) = \mathrm{ELF}_{\min}(v) + \mathrm{ELF}_{\max}(v) \tag{A.4}$$

where v denotes the vector of squared pair distances. $\mathrm{ELF}_{\min}$ penalises distances below their lower bounds: adjacent pairs below D_{min}^2, and non-adjacent pairs below $(D_{adj} + \alpha)^2$. $\mathrm{ELF}_{\max}$ penalises distances above their upper bounds: adjacent pairs above D_{adj}^2, and non-adjacent pairs above $4L^2$. The parameter α enters the non-adjacent lower bound so that as the gap increases, the loss progressively pushes non-adjacent atoms further apart. Both components use the margin ranking formulation $\mathrm{avg}(\max(0, -(v - v^t)))$ with target v^t and sign ± 1 depending on the direction of the constraint. Model weights are updated by the AdamW optimiser [152].

A.1.3 Hardware-Specific Adaptations

When deployed on real QPUs, the free-space CUDG formulation must be extended to accommodate device-specific constraints.

Aquila (QuEra): row-spacing constraint

The Aquila register is rectangular ($75\,\mu\mathrm{m} \times 76\,\mu\mathrm{m}$, up to 256 atoms). The register loading procedure organises atoms in rows, requiring adjacent rows to be spaced at least $4\,\mu\mathrm{m}$ apart in the y-direction. The DEN is adapted to enforce this constraint [11] (Figure A.2):

The rectangular domain is represented by setting $L = 75/2$ in the *CoordL* activation. A new ELF component, $\mathrm{ELF}_{\mathrm{row}}$, penalises atom pairs whose y-axis separation falls in the interval $(0, 4)\,\mu\mathrm{m}$, using a parabolic penalty centred at $2\sqrt{2}\,\mu\mathrm{m}$ combined with a threshold that zeroes the penalty when the row distance exceeds $4\,\mu\mathrm{m}$. The full loss becomes:

$$\mathrm{ELF}_{\mathrm{Aquila}}(v) = \mathrm{ELF}_{\min}(v) + \mathrm{ELF}_{\max}(v) + \mathrm{ELF}_{\mathrm{row}}(v) \tag{A.5}$$

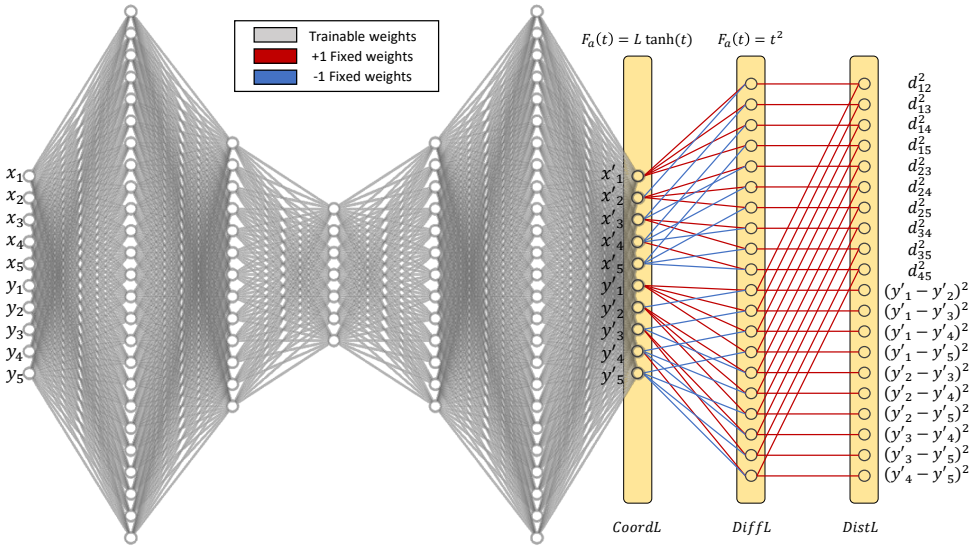


Fig. A.2 Modified DEN architecture for Aquila. The *DistL* output layer is extended to $2\binom{n}{2}$ elements: the first $\binom{n}{2}$ are full squared Euclidean distances (fed to $\mathrm{ELF}_{\mathrm{min}}$ and $\mathrm{ELF}_{\mathrm{max}}$); the remaining $\binom{n}{2}$ are squared y-axis differences (fed to $\mathrm{ELF}_{\mathrm{row}}$). Example shown for a 5-vertex graph.

A solution is declared feasible when all three components equal zero simultaneously.

Orion Alpha (PASQAL): triangular lattice remapping

The Orion Alpha register provides 61 fixed trap positions arranged in a triangular lattice with $5\,\mu$m spacing; atom positions cannot be chosen freely. The DEN is first trained under the free-space assumption ($L = 41/2$, minimum distance $5\,\mu$m, $D_{adj} = 10.26\,\mu$m) to produce a continuous-space solution. A nearest-neighbour algorithm then maps this solution to valid lattice positions:

1. Sort vertices in descending order of degree.
2. For each vertex (in sorted order), assign it to the closest unoccupied trap in the lattice.

Prioritising high-degree vertices reduces the risk of placing well-connected atoms in positions that violate the unit-disk constraint. The remapping does not guarantee strict UDG feasibility; minor adjacency violations may persist, but these proved

tolerable in practice for the BBQ-mIS use case, where the Branch-and-Bound solver remained robust to small embedding imprecisions [11].

A.1.4 Embedding Feasibility Results

General benchmark (CUDG dataset)

On a dataset of 200 graph instances ($n \in \{10, 20, \ldots, 100\}$) [10], the DEN consistently outperforms the Ipopt classical solver in feasibility rate. For $n \leq 60$, the 3D embedding variant achieves near-100% feasibility; in 2D, success rates decline from $n \approx 70$ onwards due to packing constraints. Ipopt fails to produce feasible solutions beyond $n = 20$ within comparable wall-clock budgets, even when allowed $10\times$ the DEN runtime. Average DEN runtime scales linearly with n and remains below 5 minutes per instance on a standard compute node.

QEK use case: PROTEINS dataset on Aquila

For the QEK graph classification experiment, graphs from the `PROTEINS` dataset [101] were embedded for the Aquila register. The success rate as a function of graph size is shown in Figure A.3.

For graphs with $n \leq 12$, 158 out of 207 samples (76%) were successfully embedded. For $n \leq 16$, the rate dropped to 68% (210 out of 307). The decline reflects increasing difficulty in satisfying simultaneously the minimum-distance, unit-disk, and row-spacing constraints as graph density grows. Graphs with $|K_{\max}| > 7$ or $\Delta_{\max} > 18$ are structurally unembeddable; in practice, the binding constraint for the `PROTEINS` set was the row-spacing requirement in conjunction with high local density. The adjacency gap $d_{\neg\text{adj}} - D_{adj}$ narrows with increasing n, reducing discriminability between adjacent and non-adjacent atoms, which is the primary source of hardware performance degradation described in Section 6.3.

BBQ-mIS use case: antenna network on Orion Alpha

For the graph coloring application, the 90-antenna network was decomposed into connected components; components with trivial structure (isolated vertices, complete graphs K_2–K_5) were handled analytically. The four non-trivial components

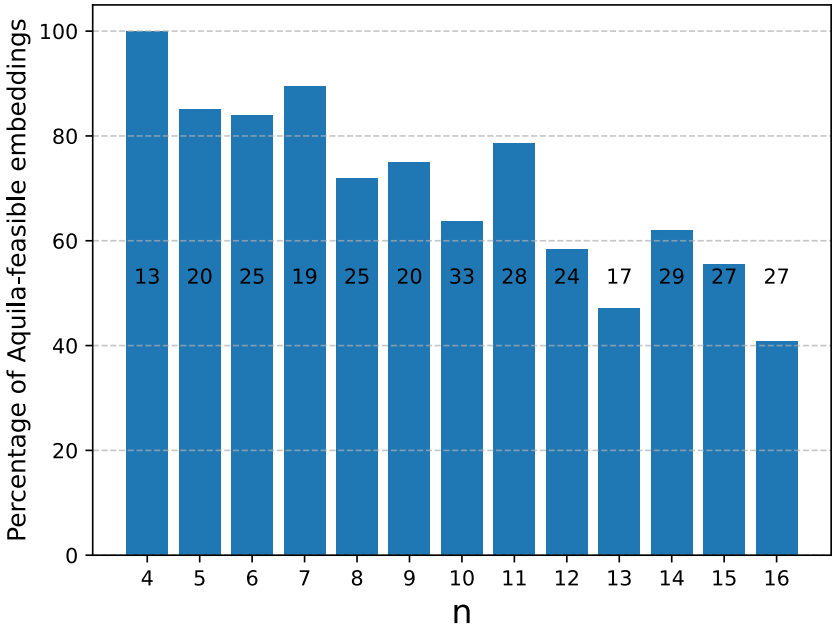


Fig. A.3 Percentage of `PROTEINS` graph samples with less than 16 nodes for which the DEN found an Aquila-feasible embedding, grouped by vertex count n. Numbers above each bar indicate the total samples in that bin.

(7, 7, 10, and 19 vertices) were embedded by the DEN followed by the nearest-neighbour lattice remapping. Under the free-space assumption, all four achieved feasible embeddings with positive adjacency gaps. After remapping to lattice coordinates, $d^{l}_{\neg\text{adj}} - D^{l}_{adj}$ was reduced or zeroed for some components, but the BBQ-mIS Branch-and-Bound solver recovered optimal graph colorings in all cases, attributing robustness to the small number of affected edges [11].

A.2 vQPU Integration within HPC Schedulers

This section documents the implementation-level details of the virtual QPU (vQPU) mechanism described in Section 5.3, covering the SLURM resource accounting interface, the quantum-partition scheduling layer, the workflow connector model, and the circuit-cutting technique used to bridge the qubit-count gap in the real-hardware vQPU experiment. Full experimental results are reported in Section 5.4; the technical material here follows the system design described in [7, 8].

A.2.1 SLURM Generic Resource and Licenses Mechanism

SLURM exposes trackable non-CPU resources through two complementary interfaces: Generic Resources (GRES) and license tokens. Both represent a countable pool of units that the scheduler deducts from when a job is admitted and restores when the job ends, without prescribing anything about the physical resource being modelled.

For QPU virtualization, the SLURM administrator registers N vQPU tokens as a GRES entry (e.g., `gres=qpu:N` in `gres.conf`) or as a license pool (e.g., `Licenses=qpu:N` in `slurm.conf`). A hybrid job requesting one token declares `--gres=qpu:1` or `--licenses=qpu:1` in its job script. SLURM then admits up to N such jobs simultaneously, multiplexing access to the single physical QPU through time sharing. The job script structure for a two-partition hybrid job under this model is shown in Listing 1 of Section 5.2.

The choice of N (the multiplexing degree) is a policy decision. Setting $N = 1$ reproduces the baseline exclusive-access model. Setting $N > 1$ enables concurrent admission of hybrid jobs while the QPU serializes circuit submissions through its

internal FIFO queue. The value of N should be calibrated to the expected quantum-to-classical workload ratio of the target application class: too large a multiplexing degree saturates the QPU queue and increases per-job wait times; too small a degree leaves the QPU underutilized. The saturation effect and the optimal operating point are characterized experimentally in Section 5.4.

A.2.2 Quantum-Partition Scheduling with HyperQueue

In the dynamic resource management experiments described in Section 5.6, the quantum subroutine (a Simulated Annealing solver acting as a proxy for a quantum annealer) must be submitted to a dedicated quantum partition on a shared SLURM cluster. Coordinating this submission from within an MPI application requires a lightweight meta-scheduler that can dispatch a job to SLURM on behalf of a running task without blocking the classical computation.

HyperQueue [8] fills this role. It runs as a server on a node allocated from the quantum partition and exposes a simple task-submission interface to worker processes. When the hybrid application reaches a quantum phase, MPI rank 0 submits the QUBO task to HyperQueue, which allocates the quantum node and executes the solver. The classical processes can optionally release their own allocations (via DMR) while the quantum task runs, or remain minimally active to enable a fast restart once the quantum result is returned.

This two-level scheduling model, SLURM for coarse-grained partition allocation and HyperQueue for fine-grained quantum task dispatch, decouples the resource-management concern from the application logic. It also preserves compatibility with the vQPU layer: if multiple hybrid jobs are running concurrently under vQPU tokens, their HyperQueue quantum submissions enter the device FIFO independently and are served in arrival order.

A.2.3 StreamFlow Quantum Connector

The workflow-based resource allocation strategy uses StreamFlow [128] as the Workflow Management System (WMS). StreamFlow exposes quantum resources through a pluggable connector architecture: adding a new quantum backend requires

extending the `Connector` base class to integrate the device's interface, whether a vendor REST API or a SLURM-managed quantum partition.

In the HPC–QC setting used in this thesis, the quantum connector mediates job submission to the SLURM quantum partition. Each time a workflow step is mapped to a quantum location, StreamFlow instantiates the connector, submits a Slurm job to the quantum partition, waits for the result, and returns the output to the next step in the workflow dependency graph. Classical resources are provisioned only for steps mapped to the CPU partition; the connector releases them between steps, implementing the separate reservation model discussed in Section 5.5.1.

The StreamFlow workflow definition follows the Common Workflow Language (CWL) standard [129], augmented with the StreamFlow topology description (mapping from workflow steps to execution locations) and an iterative loop construct for NISQ workloads, which typically execute a fixed-point iteration over the quantum–classical pipeline. This loop construct is particularly relevant for the clustering aggregation use case described in Section 5.5.3.

A.2.4 Circuit Cutting for Qubit-Scale Bridging

The real-hardware vQPU experiment dispatches QAOA circuits to the Lagrange IQM Spark quantum computer, a 5-qubit superconducting device [4]. The MIS subproblems arising from the BBQ-mIS Branch-and-Bound (Section 4.5) may involve more than 5 variables, which would exceed the hardware capacity. Circuit cutting [124] addresses this constraint by decomposing a wide circuit into smaller sub-circuits that each fit on the device, executing them independently, and recombining the results classically.

Two main cutting strategies exist. *Wire cutting* partitions the circuit by severing qubit lines and inserting measure-and-prepare operations at the cut locations. *Gate cutting* decomposes two-qubit gates into probabilistic mixtures of single-qubit operations. Both strategies incur an exponential overhead in the number of sub-circuits relative to the number of cuts applied, so they are practical only when the circuit topology admits a small number of cuts.

The MIS instances in the vQPU experiment are formulated for *(K,d)-clustered graphs*: graphs where a set of K vertex separators decomposes the remaining vertices into connected components of at most d nodes. With $d = 5$ (matching the device

qubit count) and $K = 1$ (a single central vertex separator), the full QAOA circuit contains $d + K + 1 = 7$–9 qubits and admits a single wire cut, splitting it into two 5-qubit sub-circuits. Figure 5.5 shows a concrete example. The QAOA depth is fixed at $p = 1$ to prevent the number of required cuts from increasing with depth.

The classical recombination step computes the expectation value $\langle H_C \rangle$ from the sub-circuit measurement outcomes and recovers the final MIS sample. The implementation uses the `qiskit_addon_cutting` Python library for the wire-cutting procedure; the final sampling step (not covered by the standard formulation) was implemented independently following the approach proposed in [124].

The net effect of circuit cutting on the queueing dynamics under vQPUs is that each logical quantum phase is replaced by two physical circuit submissions. This doubles the granularity of quantum events relative to an uncut circuit, increasing the opportunities for interleaving across concurrent jobs and therefore tending to improve quantum occupancy at a given multiplexing degree N.

Appendix B

Mathematical Formulations and Optimization Models

B.1 Spectral Lower Bounds on the Chromatic Number

The BBQ-mIS algorithm (Section 4.5) uses spectral lower bounds to prune the Branch-and-Bound search tree: at each node, the chromatic number of the residual induced subgraph must exceed the current incumbent coloring for the branch to be worth exploring. This section collects the spectral bounds used in the implementation and reported in Section 6.2.4.

Let $G = (V, E)$ be an undirected graph on $n = |V|$ vertices. Let A be its adjacency matrix, with eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$. Denote by $\chi(G)$, $\omega(G)$, and $\alpha(G)$ the chromatic number, clique number, and independence number of G, respectively. The complement graph is denoted $\bar{G}$.

B.1.1 Hoffman Ratio Bound

The Hoffman ratio bound [146] relates the chromatic number to the extreme eigenvalues of the adjacency matrix.

$$\chi(G) \geq 1 - \frac{\lambda_1}{\lambda_n}, \tag{B.1}$$

where λ_1 is the largest eigenvalue and λ_n is the smallest (most negative) eigenvalue. When G has no edges, $\lambda_n = 0$ and the bound is not applicable; it is most useful for graphs with negative eigenvalues, i.e., those containing at least one edge.

The bound follows from a coloring argument: if G is k-colorable, the eigenvalues of the adjacency matrix satisfy $\lambda_n \leq -\lambda_1/(k-1)$, which rearranges to give (B.1). Since $\chi(G)$ must be an integer, the effective lower bound is $\lceil 1 - \lambda_1/\lambda_n \rceil$.

B.1.2 Elphick–Wocjan σ-Bound

The Elphick–Wocjan σ-bound [147] generalises the Hoffman ratio bound by using all eigenvalues rather than only the extreme pair. Define the positive and negative spectral parts:

$$\sigma^+ = \sum_{\lambda_i > 0} \lambda_i, \qquad \sigma^- = \sum_{\lambda_i < 0} |\lambda_i|.$$

The bound is then:

$$\chi(G) \geq 1 + \frac{\sigma^+}{\sigma^-}. \tag{B.2}$$

Since $\sum_i \lambda_i = \operatorname{tr}(A) = 0$, it follows that $\sigma^+ = \sigma^-$, and (B.2) simplifies to $\chi(G) \geq 2$ whenever G has at least one edge. The bound is non-trivial in the sense that the ratio σ^+/σ^- can exceed 1 for structured graphs, producing tighter lower bounds than $\chi(G) \geq 2$.

B.1.3 Edwards–Elphick Bound

The Edwards–Elphick bound [148] combines spectral information with a combinatorial quantity. Let $q(G)$ denote the largest eigenvalue of the signless Laplacian $Q = D + A$, where D is the diagonal degree matrix. Then:

$$\chi(G) \geq \frac{1}{2} + \sqrt{\frac{1}{4} + n \cdot \frac{|E|}{n^2 - q(G)\, n}}, \tag{B.3}$$

where $|E|$ is the number of edges. This bound tends to be tighter than the Hoffman bound on graphs with high edge density, and it complements the σ-bound on sparse instances.

B.1.4 Lovász Theta Function

The Lovász theta function [149] provides a semidefinite programming (SDP) relaxation that sandwiches both the independence number and the chromatic number. For a graph G:

$$\alpha(G) \leq \vartheta(G) \leq \chi(\bar{G}), \tag{B.4}$$

where $\bar{G}$ is the complement of G. By complementation, $\vartheta(\bar{G})$ satisfies:

$$\omega(G) \leq \vartheta(\bar{G}) \leq \chi(G), \tag{B.5}$$

so $\vartheta(\bar{G})$ provides a lower bound on the chromatic number $\chi(G)$.

The theta function $\vartheta(G)$ admits the following SDP characterization:

$$\vartheta(G) = \max\left\{\langle J, X\rangle : \langle I, X\rangle = 1,\, X_{ij} = 0\ \forall (i,j) \in E,\, X \succeq 0\right\}, \tag{B.6}$$

where J is the all-ones matrix, I is the identity matrix, $\langle \cdot, \cdot \rangle$ denotes the Frobenius inner product, and $X \succeq 0$ denotes positive semidefiniteness. The dual problem is:

$$\vartheta(G) = \min\left\{\lambda_{\max}(B) : B_{ij} = 1\ \forall (i,j) \notin E \cup \{(i,i)\}\right\}, \tag{B.7}$$

where the constraint requires $B_{ij} = 1$ for all non-edges and the diagonal, and $\lambda_{\max}(B)$ is the largest eigenvalue of B.

The Lovász theta function is computable in polynomial time via interior-point SDP solvers. In the BBQ-mIS implementation, $\vartheta(\bar{G})$ is evaluated at each Branch-and-Bound node for the residual subgraph, and its ceiling $\lceil \vartheta(\bar{G}) \rceil$ is used as a lower bound on $\chi(G)$, in combination with the bounds of Sections B.1.1–B.1.3. The tightest bound among the four is adopted at each node, which ensures that the chromatic-number lower bound is as tight as the available spectral information allows.

B.1.5 Summary and Practical Notes

Table B.1 summarises the four lower bounds used in BBQ-mIS. All four are based on properties of the adjacency or Laplacian matrix and can be evaluated in polynomial time. In practice, neither bound dominates the others across all graph instances; the tightest is therefore selected at each node.

Table B.1 Spectral lower bounds on the chromatic number $\chi(G)$ used in BBQ-mIS. All are computable in polynomial time. The tightest bound is used at each Branch-and-Bound node.

Bound	**Formula**	**Key input**
Hoffman [146]	$\chi(G) \geq 1 - \lambda_1/\lambda_n$	Extreme eigenvalues of A
Elphick–Wocjan [147]	$\chi(G) \geq 1 + \sigma^+/\sigma^-$	Full spectrum of A
Edwards–Elphick [148]	Eq. (B.3)	Signless Laplacian spectrum
Lovász ϑ [149]	$\chi(G) \geq \lceil \vartheta(\bar{G}) \rceil$	SDP on complement graph

The upper bounds used to guide the search order are derived from classical greedy-coloring heuristics: the standard greedy coloring and the Welsh–Powell ordering [150]. These do not provide chromatic-number guarantees, but supply a feasible coloring quality that sets the initial incumbent and is used as the upper bound UB at each node.

B.2 BBQ-mIS Branching Formalism

This section collects the mathematical definitions and formal results underlying the Branch-and-Bound structure of BBQ-mIS, described in Section 4.5. These support the algorithm's correctness argument and define the pruning and exploration logic used in the implementation.

B.2.1 Maximal Independent Sets and the Coloring Theorem

Definition B.2.1 (Independent set)**.** Given a graph $G = (V, E)$, an *independent set* is a subset $S \subseteq V$ such that no two vertices in S share an edge in E.

Definition B.2.2 (Maximal independent set)**.** A *maximal independent set* (mIS) of $G = (V, E)$ is an independent set that is not properly contained in any other

independent set of G. Equivalently, adding any vertex not already in the set would violate independence.

Every mIS is therefore independent and inclusion-maximal, but not necessarily of maximum cardinality. The following theorem, cited in [6], is the theoretical foundation of the BBQ-mIS branching strategy.

Theorem 1 (Optimal graph coloring via mIS)**.** Every graph $G = (V,E)$ admits an optimal coloring in which at least one color class is a maximal independent set.

Proof. Let $C = \{C_1, \ldots, C_k\}$ be an optimal coloring of G, where $k = \chi(G)$ and each C_i is an independent set. Let I be any mIS of G such that $C_1 \subseteq I$ (such an I always exists because C_1 is an independent set and any independent set can be extended to a mIS by greedily adding non-adjacent vertices). Define the coloring

$$C' = \{I,\, C_2 \setminus I,\, \ldots,\, C_k \setminus I\}.$$

Each color class in C' is still independent, since removing vertices from an independent set preserves independence, and I is independent by assumption. Every vertex in V remains covered: vertices in I are assigned the first color, and vertices in $C_i \setminus I$ for $i \geq 2$ are assigned color i. No vertex is left uncolored, because any vertex not in I must belong to some C_i with $i \geq 2$ (otherwise it would be in $C_1 \subseteq I$, a contradiction). Therefore $|C'| \leq |C| = k$, and since $k = \chi(G)$ is minimal, C' is also optimal. The first color class of C' is the mIS I. □

Theorem 1 allows the Branch-and-Bound search to restrict its branching candidates to mISs. Any non-maximal independent set returned by the MIS solver is discarded, because the theorem guarantees it cannot lead to an optimal coloring that the maximal-extension would not cover.

B.2.2 Branch-and-Bound Node State and Pruning

At each node of the BBQ-mIS Branch-and-Bound tree, the state is represented by a tuple

$$(H, C, k, LB, UB),$$

where:

- $H = (U, I)$ is the induced subgraph remaining after removing the mIS assigned in the parent branch;
- C is the partial feasible coloring assembled so far, extended with a worst-case per-vertex assignment for the remaining vertices in H;
- k is the current number of colors in C;
- LB is a lower bound on the chromatic number achievable from this node, computed as $LB = d + \lceil \chi_{\text{lb}}(H) \rceil$, where d is the depth of the node in the tree and $\chi_{\text{lb}}(H)$ is the tightest spectral lower bound from Section B.1;
- $UB = \min(\lceil UB_G \rceil, \lceil UB_{WP} \rceil)$ is an upper bound on $\chi(H)$, used to guide the exploration order.

Three pruning rules reduce the number of nodes generated or explored.

Pruning by non-improvement. A node is discarded when $LB \geq k^*$, where k^* is the best (smallest) coloring found so far. Exploring it cannot improve the incumbent.

Pruning by unfeasibility. A candidate mIS that is either non-independent (adjacent vertices present) or non-maximal (can be extended) is rejected before generating a child node. This directly enforces the requirement of Theorem 1.

Pruning by redundancy. Two different branching paths can produce the same induced subgraph $H = (U, I)$ by removing the same vertex set in a different order. To avoid exploring identical subgraphs twice, each H is associated with a fingerprint

$$\text{fp}(H) = \sum_{u \in U} 2^u - 1, \tag{B.8}$$

where vertices are indexed by integer labels. A new node is generated only if its fingerprint has not yet been encountered during the search.

B.2.3 Exploration Priority Score

To balance depth-first and best-first exploration, BBQ-mIS assigns each queued node a priority score

$$s(H, UB) = -UB \times |I|, \tag{B.9}$$

where $|I|$ is the number of edges in the remaining subgraph H and UB is the upper bound at that node. Nodes with higher s are explored first.

The rationale is the following. At the top of the tree, induced subgraphs are large and densely connected, so $|I|$ is large and priority scores are low. The exploration therefore proceeds depth-first in the early stages, yielding an initial feasible coloring quickly, similar to Greedy-it-MIS. As the search progresses, the algorithm shifts toward nodes with smaller $|I|$ combined with tighter UB, favoring branches where the remaining coloring problem is both simpler and more tightly constrained. This adaptive balance reduces the number of nodes explored before reaching the optimal solution.

Appendix C

Additional Experimental Results

C.1 QRC Credit-Default Pipeline: Supplementary Details

This section provides supplementary implementation details and experimental results for the Quantum Reservoir Computing credit-card default prediction case study reported in [1]. Full methodology is described in Section 6.2.2 and main classification results in Section 6.3.1.

C.1.1 Dataset Preprocessing Details

After data cleaning, the 24-feature representation of the `CARDS_30000` dataset is reduced to 12 aggregated features via a PCA-based hierarchical clustering procedure. Agglomerative clustering is applied to the 24 features in the principal-component score space, with a threshold set to half the maximum pairwise distance; this yields the 12 clusters listed in Table C.1. Features within each cluster are averaged to form a single representative value.

The three datasets considered in the study are class-imbalanced, with a defaulter fraction of approximately 22% consistently across all subsets. Table C.2 summarises the training-set sizes produced by each resampling strategy for the three dataset scales; validation and test splits are left unmodified in all cases.

Table C.1 Feature clusters produced by agglomerative clustering in PCA score space on `CARDS_30000`. Each cluster becomes one input feature for the QRC layer after averaging.

Cluster	Features
0	`PAY_3`, `PAY_4`
1	`PAY_5`, `PAY_6`
2	`PAY_AMT1`, ..., `PAY_AMT6`
3	`MARRIAGE_1`
4	`BILL_AMT1`, ..., `BILL_AMT6`
5	`MARRIAGE_2`
6	`PAY_1`
7	`LIMIT_BAL`
8	`AGE`
9	`EDUCATION`
10	`PAY_2`
11	`SEX_1`

Table C.2 Training-set class distributions after each resampling strategy, for the three dataset scales. Validation and test sets are held fixed (class-1 counts in parentheses). CC = Cluster Centroids; K-SMOTE = K-Means SMOTE.

Dataset	Split	None	SMOTE	K-SMOTE	CC
`CARDS_30000`	Train total	20 720	32 194	32 198	9 246
	Class 1	4 623	16 097	16 101	4 623
	Val / Test	4 440 / 4 441 (991 class 1 each)			
`CARDS_2500`	Train total	1 799	2 794	2 797	804
	Class 1	402	1 397	1 400	402
	Val / Test	386 / 386 (86 class 1 each)			
`CARDS_1000`	Train total	700	1 088	1 091	312
	Class 1	156	544	547	156
	Val / Test	150 / 150 (34 class 1 each)			

C.1.2 Aquila Register Embedding

The classical emulation of the neutral-atom reservoir assumes atoms are arranged along a one-dimensional chain. On Aquila, however, atoms must be placed within a two-dimensional register of size $75\,\mu\text{m} \times 125\,\mu\text{m}$, with a minimum inter-atom vertical separation of $4\,\mu\text{m}$. The 1D chain is therefore mapped onto the 2D plane using one of two strategies depending on chain length:

- *Column embedding*: if the chain fits within the register height, atoms are placed in a single vertical column (equivalently, a single row after a 90° rotation).
- *Serpentine embedding*: if the chain exceeds the register height, it wraps into a snake-like pattern alternating column directions.

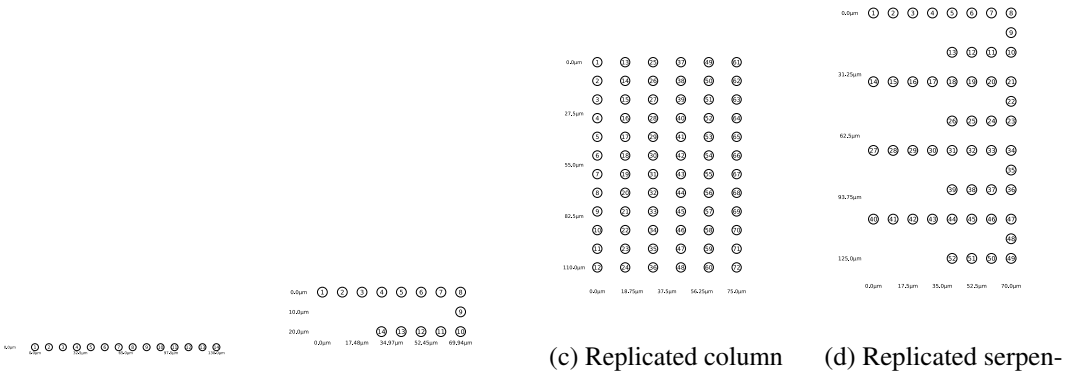
To improve statistical sampling without increasing pulse count, multiple independent replicas of the encoded atom array are placed within the same register. Replicas are separated by at least $15\,\mu\text{m}$, chosen to be greater than both the Rydberg blockade radius and the nominal inter-atom spacing $r_0 = 10\,\mu\text{m}$, so cross-replica interactions are negligible. For position encoding, replicas are arranged by row rather than by column to ensure that the $4\,\mu\text{m}$ vertical-spacing constraint is met between neighbouring rows. The four resulting layout strategies are illustrated in Figure C.1.

For both encoding strategies and all dataset scales considered, the 12-atom (or 13-atom for position encoding) array fits within a single column of the available register height, so the column embedding is used throughout with $N_{\text{repl}} = 6$ replicas.

C.1.3 Feature Correlation Analysis

To quantify the agreement between Aquila-generated reservoir features and their shot-based emulation counterpart, and to select the optimal number of measurements n_{opt}, a statistical correlation analysis was performed on ten samples randomly drawn from `CARDS_2500` using the detuning encoding [1].

For each sample, quantum simulations were run on Aquila with up to 1000 measurements per timestep. The Pearson correlation coefficient between Aquila QRC features and the corresponding shot-based emulation features was computed as a function of both the number of shots and the timestep index (1 through 5). Figure C.2 reports the resulting correlation maps.



(a) Single column (b) Serpentine (c) Replicated column ($\times 6$) (d) Replicated serpentine ($\times 4$)

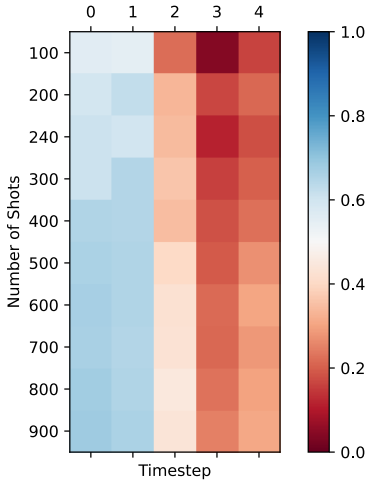
Fig. C.1 Aquila register embedding strategies for a 1D atom array. (a) Single-column layout; (b) serpentine layout for longer chains; (c) replicated column with six copies; (d) replicated serpentine with four copies. Cross-replica separation of $15\,\mu$m suppresses inter-replica Rydberg interactions.

Three observations emerge. First, correlation is not uniform across timesteps: the earliest timestep retains the strongest agreement with the emulated reference, while later timesteps show progressively larger discrepancy. This is consistent with the physical picture of decoherence accumulating over the analog evolution duration. Second, increasing the shot count beyond approximately 200 does not substantially improve correlation, as shown by the saturation in the mean-correlation plot. Third, the degradation at the last timestep (t_5) is markedly more severe than at the first, highlighting that the final components of the reservoir output vector are the most noise-contaminated. Based on this analysis, $n_{\text{opt}} = 240$ total shots per sample was selected, corresponding to 40 shots per timestep across $N_{\text{repl}} = 6$ register replicas.

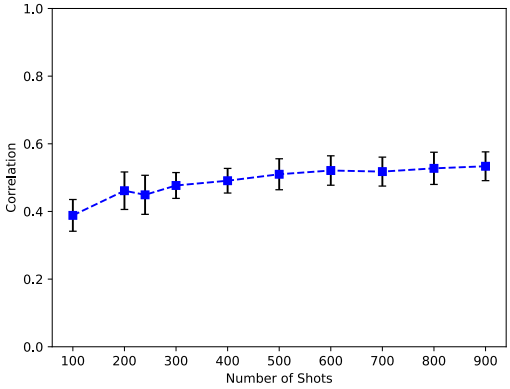
C.1.4 DNN Architecture and Hyperparameters

A Deep Neural Network is used as the nonlinear classical comparator for the QRC pipeline. Unlike the reservoir-based classifiers, the DNN operates directly on the 12 preprocessed features (before the reservoir layer). Its architecture, implemented in PyTorch [139], is summarised in Table C.3.

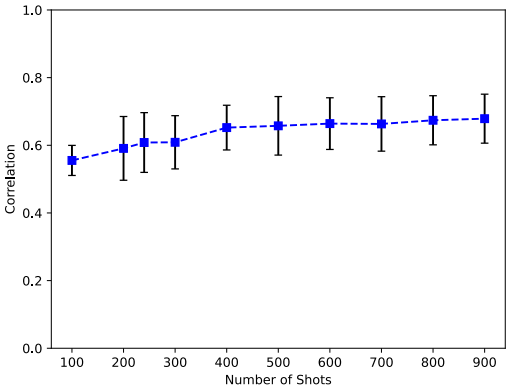
The model is trained for up to 1000 epochs using the Adam optimiser with cross-entropy loss. Early stopping is applied with a patience of 30 epochs to prevent overfitting. Hyperparameter tuning uses a grid search over learning rate



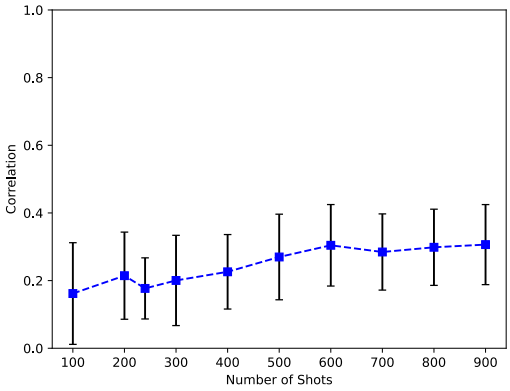
(a) Correlation as a function of shots and timestep (10 samples).



(b) Mean across timesteps.



(c) First timestep (t_1).



(d) Last timestep (t_5).

Fig. C.2 Statistical correlation between Aquila QRC features (detuning encoding) and shot-based emulation, as a function of the number of shots and timestep index, for ten randomly selected `CARDS_2500` samples. Correlation degrades at later timesteps due to decoherence accumulation and saturates beyond ≈ 200 shots per timestep.

Table C.3 DNN architecture used as the classical nonlinear baseline. Input dimensionality $f = 12$ (preprocessed features). All hidden layers include dropout regularization. The output layer uses softmax activation for binary classification.

Layer	Size	Activation	Notes
Input	$f = 12$	—	preprocessed features
FC	64	ReLU	dropout
FC	128	ReLU	dropout
FC	256	ReLU	dropout
FC	128	ReLU	dropout
FC	64	ReLU	dropout
Output	2	Softmax	binary classification

$\eta \in \{0.1, 0.01, 0.001\}$ and batch size $b \in \{64, 128, 256\}$, selecting the configuration that maximises the F1-score on the validation set.

C.2 QEK Graph-Classification Pipeline: Supplementary Details

This section provides supplementary implementation details for the Quantum Evolution Kernel graph-classification case study reported in [5]. Full methodology is described in Section 6.2.3 and main classification results in Section 6.3.

C.2.1 PROTEINS Dataset Statistics

The full `PROTEINS` benchmark comprises 1,113 graphs (663 enzymes, 450 non-enzymes), with statistics summarised in Table C.4. The node and edge distributions are highly skewed: most graphs contain fewer than 100 nodes, but the maximum is 620. Figure C.3 shows the full distribution.

After applying the node-count filter ($n \leq 256$ for `PROTEINS256`; $n \leq 12$ for `PROTEINS12`) and the unit-disk feasibility filter via the DEN embedding procedure (see Appendix A.1), the two working subsets are obtained. The resulting class compositions are those reported in Table 6.2: `PROTEINS12` is strongly imbalanced (23% minority class), while `PROTEINS256` is less skewed (39% minority class). This difference is primarily a consequence of which graph sizes admit feasible UD

Table C.4 Overall statistics of the PROTEINS dataset before filtering.

Statistic	Value
Total graphs	1,113
Class 1 (enzymes)	663
Class 2 (non-enzymes)	450
Min nodes	4
Avg nodes	39.5
Max nodes	620
Min edges	5
Avg edges	72.81
Max edges	1,049

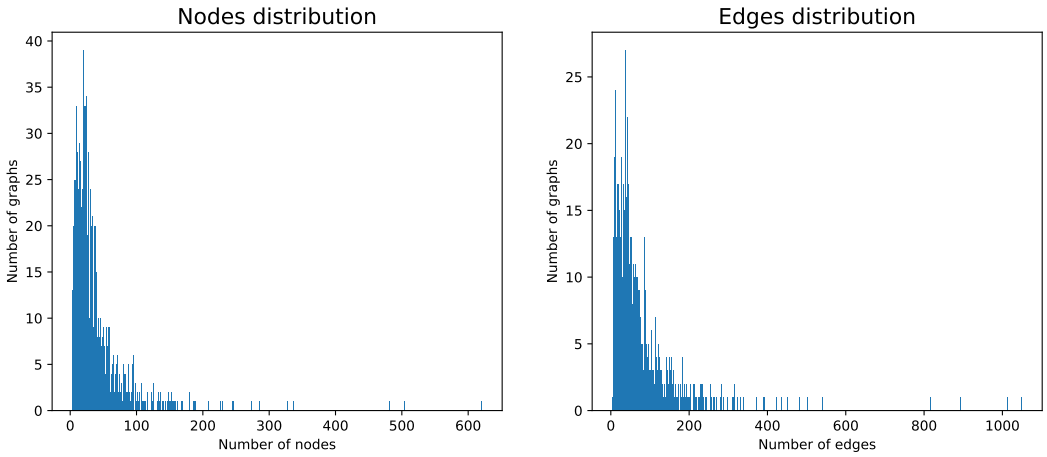


Fig. C.3 Node and edge distributions across the full PROTEINS dataset. Most graphs contain fewer than 100 nodes. The vertical dashed line at 12 nodes marks the boundary of PROTEINS12, used for classical emulation and pulse optimization; the boundary of PROTEINS256 at 256 nodes encompasses nearly the full dataset by count.

embeddings: the DEN model has higher success rates on smaller, sparser graphs, and those tend to be non-enzymes.

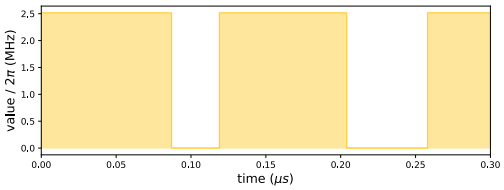
C.2.2 Waveform Comparison: Λ_a versus Λ_{BO}

The QEK pulse schedule is parameterised as $\Lambda = \{\tau_0, t_0, \tau_1, t_1, \tau_2\}$, where the τ_i are the durations of the three Rabi-drive segments (corresponding to mixing-Hamiltonian rotations) and the t_i are the two free-evolution intervals. The Rabi amplitude is fixed to $\Omega_0 = 15.8\,\mathrm{rad}/\mu\mathrm{s}$, the maximum supported by Aquila, in order to minimise total evolution time and reduce decoherence. The rotation angle for each driven segment is then $\theta_i = \Omega_0\,\tau_i$.

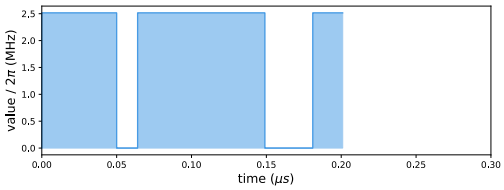
Two parameter sets are compared throughout the case study. The first, Λ_a, is the reference schedule from [98], which the present work validated as a starting point. The second, Λ_{BO}, is obtained by Bayesian Optimization on the `PROTEINS12` subset (see Section C.2.3). Table C.5 summarises the key parameters; Figure C.4 shows the corresponding pulse waveforms.

Table C.5 Comparison of the two QEK pulse parameter sets. Rotation angles are computed as $\theta_i = \Omega_0\,\tau_i$ with $\Omega_0 = 15.8\,\mathrm{rad}/\mu\mathrm{s}$. The Λ_{BO} schedule is 37% shorter in total duration, reducing decoherence exposure.

Parameter set	τ_0	t_0	τ_1	t_1	τ_2	**Total**	θ_1
Λ_a	87 ns	–	84 ns	–	72 ns	317 ns	$\approx 80^\circ$
Λ_{BO}	50 ns	–	85 ns	–	20 ns	201 ns	$\approx 80^\circ$



(a) Reference schedule Λ_a [98]. Total duration: 317 ns.



(b) Bayesian-optimized schedule Λ_{BO}. Total duration: 201 ns.

Fig. C.4 Pulse waveforms for the two QEK parameter sets. Both consist of three Rabi-drive segments (mixing Hamiltonian $\hat{H}_\theta$) separated by free-evolution intervals (Ising Hamiltonian $\hat{H}_G$). The optimized schedule Λ_{BO} is characterized by a dominant central pulse ($\approx 80^\circ$ rotation) flanked by two shorter pulses ($\approx 45^\circ$ and $\approx 18^\circ$), compared to the more uniform pulse durations of Λ_a.

The key qualitative difference between the two schedules is the distribution of rotation angle across the three pulses. In Λ_a, all three driven segments have similar durations (87 ns, 84 ns, 72 ns), producing a roughly uniform sequence of rotations. In Λ_{BO}, the central pulse dominates ($\tau_1 = 85\,\mathrm{ns}$, $\theta_1 \approx 80°$), while the flanking pulses are shorter ($\tau_0 = 50\,\mathrm{ns}$, $\theta_0 \approx 45°$; $\tau_2 = 20\,\mathrm{ns}$, $\theta_2 \approx 18°$). The 116 ns reduction in total duration reduces decoherence exposure and brings emulated and hardware energy distributions into closer agreement.

The effect of the two parameter sets on the energy distribution is illustrated in Figure C.5, which shows the histogram obtained from the same graph under each schedule. The optimized schedule produces a distribution with stronger contrast between low- and high-energy configurations, which translates into more discriminative pairwise Jensen–Shannon divergences and hence a more informative kernel matrix.

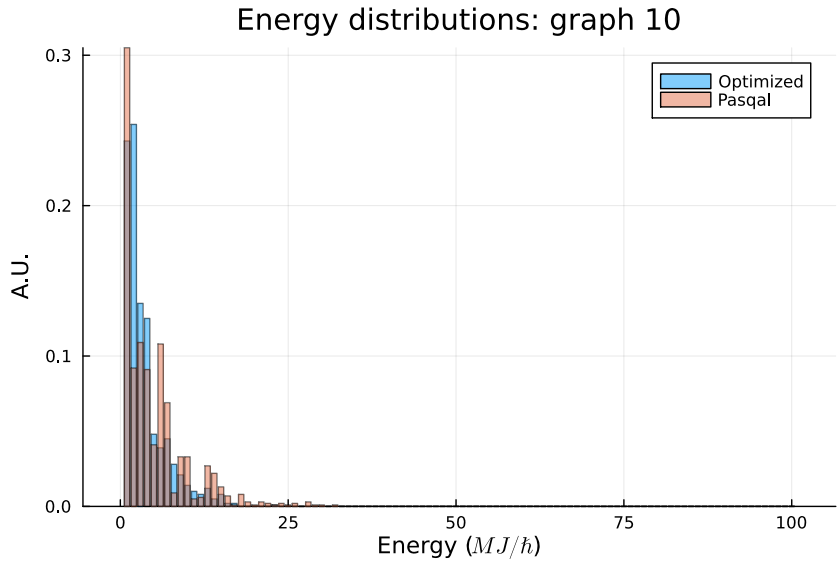


Fig. C.5 Comparison of energy distributions produced by Λ_a (light red histogram labeled as "Pasqal") and Λ_{BO} (blue histogram labeled as "Optimized") on the same graph from `PROTEINS12`. The optimized schedule yields a distribution with greater separation between energy levels, producing a more discriminative kernel.

C.2.3 Bayesian Optimization Configuration

The pulse parameters Λ_{BO} are obtained by maximizing the average cross-validated F1-score on `PROTEINS12` through Bayesian Optimization (BO). Since each evalua-

tion of the objective requires computing the full quantum evolution, constructing the 143×143 kernel matrix, and running 10-fold cross-validated SVM training with a grid search over C and w, direct gradient-based or random-search optimization would be impractical. BO is well suited for this regime because it models the objective as a sample from a Gaussian Process and selects the next evaluation point to maximize expected improvement.

Surrogate model. The surrogate is a Gaussian Process $\mathscr{GP}$ with:

- Mean: $\mu_0 \sim \mathscr{N}(0, 10)$ (zero-mean with standard deviation 10).
- Covariance: isotropic $\frac{5}{2}$-Matérn kernel,

$$k(x,x') = \sigma^2 \left(1 + \frac{\sqrt{5}\,|x-x'|}{\ell} + \frac{5\,|x-x'|^2}{3\ell^2}\right) \exp\left(-\frac{\sqrt{5}\,|x-x'|}{\ell}\right),$$

  with length scale $\ell = 10$ and signal standard deviation $\sigma = 10$.
- Kernel hyperparameters (ℓ and σ) are updated every 10 iterations via Maximum A Posteriori (MAP) estimation.

Acquisition function. The Upper Confidence Bound (UCB) is used:

$$\alpha(x) = \mu(x) + k\,\sigma(x),$$

where k controls the exploration–exploitation trade-off ($k = 1$ in the reported experiments).

Parameter constraints. The following constraints are enforced on $\Lambda_{\mathrm{BO}} = \{\tau_0, t_0, \tau_1, t_1, \tau_2\}$:

$$\tau_0 + t_0 + \tau_1 + t_1 + \tau_2 < 500\,\mathrm{ns} \tag{C.1}$$

$$t_i > 5\,\mathrm{ns}, \quad i \in \{0,1\} \tag{C.2}$$

$$\tau_i > 5\,\mathrm{ns}, \quad i \in \{0,1,2\} \tag{C.3}$$

Constraint (C.1) limits total evolution time to reduce decoherence. Constraints (C.2) and (C.3) reflect the finite bandwidth of the optical components, which prevents

arbitrarily short pulses. The maximum number of BO iterations is set to 50, which proved sufficient for convergence on the five-dimensional parameter space.

C.3 BBQ-mIS: Supplementary Benchmark Results

This section provides supplementary experimental results for the BBQ-mIS graph-coloring case study reported in [6]. Full methodology is described in Section 4.5 and Section 6.2.4; the real-hardware evaluation on Aquila is reported in Section 6.4. The results here cover two aspects not included in the main chapters: emulation runtimes on the validated small-graph benchmark and a large-graph scaling study using a classical MIS surrogate.

C.3.1 Emulation Runtimes on the Small-Graph Benchmark

The emulation-based validation campaign described in Section 6.2.4 used the Pulser library to simulate the quantum MIS solver on 120 unit-disk graph instances with $n \in \{10, 11, 12, 13, 14, 15\}$. Table C.6 reports the minimum, average, and maximum wall-clock times for Greedy-it-MIS and BBQ-mIS across each graph size, obtained on an IBM Power9-based cluster (32 cores/node, 256 GB/node) using up to 16 MPI processes.

Table C.6 Wall-clock emulation time (hh:mm:ss) for Greedy-it-MIS and BBQ-mIS on the small-graph benchmark (Pulser-based quantum MIS solver). Times increase with n due to the exponential cost of state-vector simulation. BBQ-mIS is slower than Greedy-it-MIS because it invokes the MIS solver at multiple Branch-and-Bound nodes; the overhead diminishes in relative terms as the first (largest) MIS call dominates.

	Greedy-it-MIS			**BBQ-mIS**		
n	Avg	Min	Max	Avg	Min	Max
10	00:06:19	00:04:21	00:07:55	00:41:27	00:10:46	01:17:55
11	00:11:51	00:04:53	00:18:49	00:59:23	00:27:25	01:31:20
12	00:11:23	00:05:43	00:17:25	04:23:51	00:05:43	08:41:59
13	00:34:34	00:27:18	00:41:50	05:07:57	00:15:31	10:00:23
14	01:00:58	00:18:41	01:43:15	05:44:45	02:38:09	08:51:22
15	02:29:23	01:21:48	03:36:59	08:48:57	00:39:50	11:03:14

The runtime growth from $n = 10$ to $n = 15$ is steep for both algorithms, reflecting the exponential state-space of Pulser state-vector simulation rather than the combinatorial cost of the coloring search. The large variance in BBQ-mIS times at $n = 12$ and $n = 13$ is consistent with instance-dependent branching depth: graphs that happen to have simple coloring structure terminate early, while graphs with denser connectivity require more Branch-and-Bound nodes and therefore more MIS solver calls.

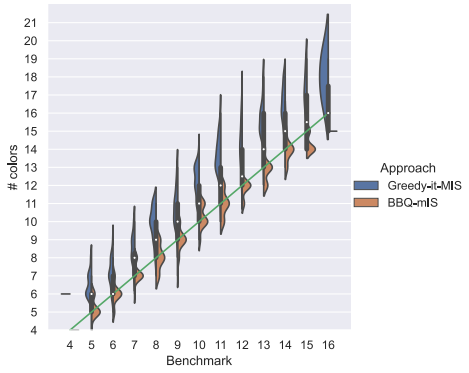
C.3.2 Large-Graph Benchmark with Classical MIS Surrogate

To assess BBQ-mIS on larger graphs beyond the emulation feasibility limit, the quantum MIS solver was replaced by a classical NetworkX heuristic [109], which provides maximal independent sets in polynomial time. This surrogate allows the Branch-and-Bound logic to be evaluated on graph sizes inaccessible to Pulser emulation and provides a reference for future hardware scaling.

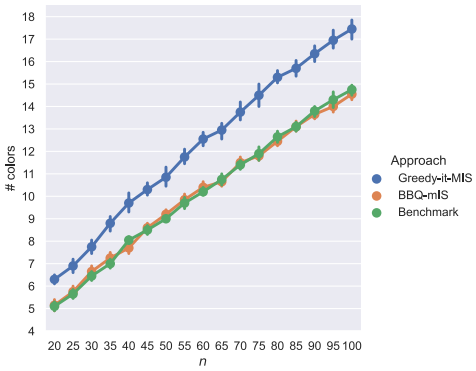
The large-graph dataset consists of unit-disk graphs with $n \in \{20, 30, \ldots, 100\}$, with 20 samples per size. On this dataset, the BBQ-mIS maximum node limit of 50 is always reached, so the returned coloring represents the best solution found within the budget rather than a certified optimum. The Gurobi classical solver provides the exact chromatic-number reference up to $n = 20$; for larger sizes, the reference is the best known solution from the classical benchmark.

Figure C.6a groups results by the benchmark coloring count. Figure C.6b shows performance as a function of n.

The key observation is that BBQ-mIS, even when capped at 50 Branch-and-Bound nodes, produces colorings that match or improve upon the benchmark across all graph sizes and benchmark classes. Greedy-it-MIS, by contrast, consistently requires more colors than necessary regardless of n. This confirms that the Branch-and-Bound scaffolding, rather than the MIS solver technology, is the primary driver of solution quality. In the context of this thesis, the implication is that the quantum MIS subroutine and a classical surrogate are functionally interchangeable within the BBQ-mIS framework, as long as the returned sets are maximal independent sets. The validated coloring logic therefore remains applicable when quantum hardware is available, with hardware-specific quality depending on the accuracy and diversity of the mIS histogram rather than on the coloring algorithm itself.



(a) Results grouped by the benchmark coloring count. Distributions of BBQ-mIS and Greedy-it-MIS solutions are shown for each benchmark class.



(b) Results as a function of graph size n. At all sizes, Greedy-it-MIS produces consistently worse colorings than the benchmark.

Fig. C.6 Graph coloring results on the large-graph dataset ($n \in \{20, \ldots, 100\}$) using a classical MIS surrogate in place of the quantum solver. BBQ-mIS solutions are at least as good as the benchmark across all tested instances. Greedy-it-MIS consistently returns colorings with more colors than necessary.

C.4 Resource Management: Supplementary Experimental Results

This section provides supplementary experimental results for the vQPU, malleability, and workflow resource management strategies described in Chapter 5. The main results and their interpretation are discussed in Sections 5.4 and 5.6. The data here supplement those sections with two additional elements: the production-scale results from the Leonardo EuroHPC system (not reported in the main text) and node-occupancy profiles that contextualise the variance observed in the Leonardo experiments.

C.4.1 Leonardo Production Results: Dynamic Allocation Strategies

The Qluster experiments reported in Section 5.6 were conducted on a controlled testbed with no external user load, yielding low-variance results. The Leonardo experiments extend the evaluation to a production EuroHPC Tier-0 system [8], where competing workloads from other users introduce scheduling delays and resource

contention that are representative of real operational conditions. The experimental setup follows that of Section 5.6.1: three scheduling configurations (baseline, DMR malleability, StreamFlow workflow) are compared on the clustering aggregation use case under two quantum time-scale proxies (two-minute-long quantum jobs, and sub-second quantum jobs).

Table C.7 reports wall time and classical resource usage (node-seconds) on Leonardo with two-minute-long quantum jobs. Consistent with the Qluster results, the baseline approach is the fastest in wall time but the most wasteful in resources. DMR reduces resource consumption by up to 45.7% relative to the baseline, at the cost of a modest increase in wall time (up to 5%) due to reconfiguration overheads. StreamFlow achieves the lowest resource consumption (up to 64% fewer node-seconds than the baseline), with wall time comparable to the other approaches within the observed variance.

Table C.7 Leonardo: executions with two-minute-long quantum jobs (10 valid runs out of 32 scheduled). Wall time and node-second resource usage are averaged over valid runs; standard deviations reflect multi-tenant cluster conditions.

Execution type	**Wall time [s]**	**Resource usage [node-s]**
Baseline	1126.10 ± 21.20	3378.30 ± 63.59
DMR (malleability)	1151.50 ± 40.63	1835.60 ± 41.74
Workflow (StreamFlow)	1127.90 ± 20.93	1216.80 ± 6.55

Table C.8 reports the corresponding results for sub-second quantum jobs. In this regime, the quantum phase contributes negligibly to total wall time, and the opportunity for resource reclamation during quantum execution is minimal. Accordingly, all three strategies converge in wall time, while resource usage remains reduced for DMR and StreamFlow relative to the baseline due to savings during the serial fraction of the classical phase. The substantially higher standard deviations compared to the medium-job case reflect elevated cluster load during the short-job experiments.

The large variance in Table C.8, particularly for DMR, is attributable to the state of the Leonardo cluster during the experimental period. In several runs, Slurm delays in reporting node availability introduced a fixed artificial overhead to each DMR expanding reconfiguration (approximately 2 seconds per reconfiguration), which becomes significant when the quantum phase is short. Newer DMR versions detect node readiness dynamically and may reduce this overhead.

Table C.8 Leonardo: executions with short (< 1 second) quantum jobs (29 valid runs out of 32 scheduled). High standard deviations reflect multi-tenant scheduling interference during the experimental period.

Execution type	Wall time [s]	Resource usage [node-s]
Baseline	803.69 ± 516.77	2411.07 ± 1550.30
DMR (malleability)	844.03 ± 529.89	1517.28 ± 538.12
Workflow (StreamFlow)	818.48 ± 420.97	1241.34 ± 22.79

C.4.2 Node Occupancy on Leonardo during the Experimental Campaign

Figures C.7 and C.8 show the node occupancy profiles of the Leonardo DCGP partition (used for the experiments) and the Booster partition respectively, sampled at two-hour intervals during the week of 13–17 November 2024.

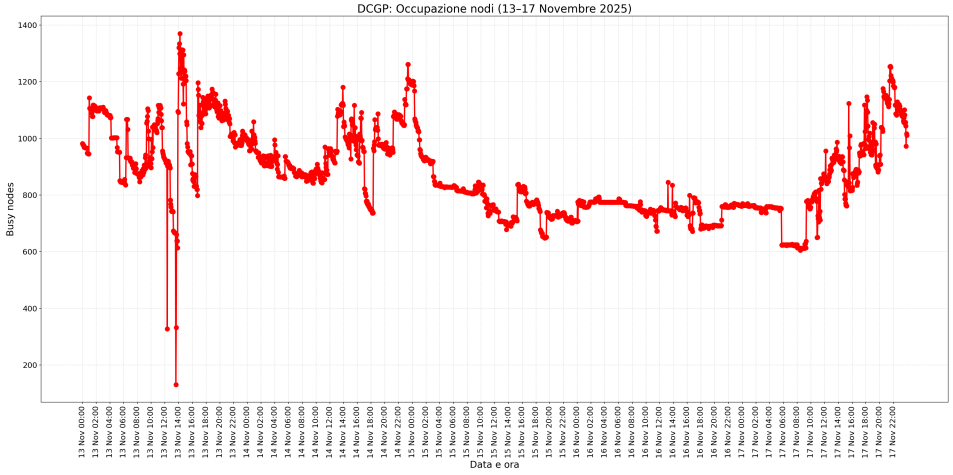


Fig. C.7 Node occupancy on the Leonardo DCGP partition over the experimental period (13–17 November 2024, 2-hour resolution). The cluster operates at near-full capacity for most of the period, with short troughs during off-peak hours. This persistent high load is the primary driver of the scheduling delays and elevated variance observed in the Leonardo experiments (Tables C.7 and C.8).

The occupancy data confirm that Leonardo operated under near-saturation load throughout the experimental campaign. This context is important for interpreting the results: the observed variance does not reflect instability in the scheduling strategies themselves, but rather the interference of concurrent user workloads, which affected Slurm queue times and node allocation delays in a non-deterministic fashion. The

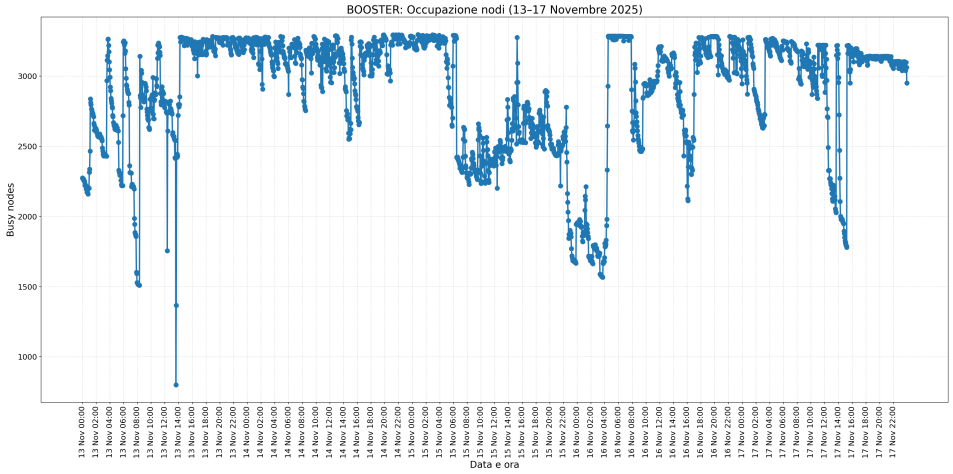


Fig. C.8 Node occupancy on the Leonardo Booster partition over the same period. The Booster partition (GPU-accelerated) shows a similar high-utilization pattern, confirming that Leonardo operated under substantial multi-tenant load throughout the experimental campaign.

Qluster results (Tables 5.1 and 5.2 in Section 5.6) provide a lower-variance baseline under controlled conditions, from which the same qualitative trends are more cleanly visible.