

THESIS FOR THE DEGREE OF LICENTIATE OF ENGINEERING

Classical Computing as an Enabler for Quantum Computing: From Scalable Exploration to Reliable Operation

ALESSIO CICERO

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY AND UNIVERSITY OF
GOTHENBURG
Gothenburg, Sweden, 2026

Classical Computing as an Enabler for Quantum Computing: From Scalable Exploration to Reliable Operation

ALESSIO CICERO

Supervisor:

*Professor Pedro Petersen Moura Trancoso, Chalmers University of Technology
and University of Gothenburg*

Co-Supervisors:

*Professor Anton Frisk Kockum, Chalmers University of Technology
Professor Mats Granath, University of Gothenburg*

Examiner:

*Professor Ioannis Sourdis, Chalmers University of Technology and University
of Gothenburg*

Discussion Leader:

Associate Professor Alexandru Paler, Aalto University

© Alessio Cicero, 2026
except where otherwise stated.
All rights reserved.

Department of Computer Science and Engineering
Division of Computer and Network Systems
Chalmers University of Technology and University of Gothenburg
SE-412 96 Göteborg,
Sweden

Printed by Chalmers Digitaltryck,
Gothenburg, Sweden 2026.

*“The key conceptual point we have grasped is that we can fight
entanglement with entanglement.”*

- John Preskill, *Reliable Quantum Computers* (1998)

Classical Computing as an Enabler for Quantum Computing: From Scalable Exploration to Reliable Operation

ALESSIO CICERO

*Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg*

Abstract

Quantum computing may solve selected classes of problems more efficiently than classical computers, with applications in areas such as cryptography, chemistry, and optimization. However, current quantum systems remain limited in scale and reliability and depend on substantial classical-computing support. This thesis investigates this challenge in two complementary contexts: classical simulation, where efficient computation enables the exploration of larger quantum systems; and quantum error correction, where efficient real-time decoding enables more reliable quantum operation.

The first part examines the classical simulation of quantum computers, focusing on the computational and memory challenges that limit its scalability. It organizes the field across device, gate, and algorithmic levels, with particular attention to state-vector, path-based, hybrid, and tensor-based methods. The survey relates their computation, memory, data-movement, and communication bottlenecks to acceleration approaches for CPUs, GPUs, FPGAs, and distributed systems. It further highlights recurring optimization strategies, including reduced precision, data compression, parallel execution, improved memory locality, specialized matrix units, and reconfigurable hardware, providing a common framework for current techniques and future acceleration opportunities.

The second part addresses the real-time classical processing required for quantum error correction. Starting from a high-accuracy Graph Neural Network surface-code decoder, it applies algorithm-hardware co-design through hardware-guided pruning and retraining, input-graph filtering, quantization, and a custom FPGA architecture. For code distances up to $d = 7$, the resulting designs satisfy both max-latency and average-latency decoding requirements while reducing the logical error rate by 13% and 40%, respectively, compared with the evaluated Minimum Weight Perfect Matching baseline.

Together, these works show how classical computer-architecture techniques can support quantum computing from scalable exploration through simulation to reliable operation through real-time error correction.

Keywords

quantum computing, acceleration, quantum error correction, quantum computers simulation

List of Publications

Appended publications

This thesis is based on the following publications:

- [**Paper I**] **Alessio Cicero**, Mohammad Ali Maleki, Muhammad Waqar Azhar, Anton Frisk Kockum, and Pedro Trancoso, *Simulation of quantum computers: Review and acceleration opportunities*.
ACM Transactions on Quantum Computing 7, no. 1 (2025): 1-35.
- [**Paper II**] **Alessio Cicero**, Luigi Altamura, Moritz Lange, Mats Granath, and Pedro Trancoso, *Low Latency GNN Accelerator for Quantum Error Correction*.
To appear in the proceedings of the 2026 IEEE International Conference on Quantum Computing and Engineering (QCE).

Other publication

The following publication was published during my PhD studies. However, it is not appended to this thesis, due to contents not related to this thesis.

[**Paper I**] Altamura, Luigi, **Alessio Cicero**, Mateo Vázquez Maceiras, Mohammad Ali Maleki, and Pedro Trancoso, *SISA: A Scale-In Systolic Array for GEMM Acceleration*. *European Conference on Parallel Processing (pp. 359-372)*. Cham: Springer Nature Switzerland.

Acknowledgment

I would like to thank my supervisor, Professor Pedro Trancoso, for his guidance and support, and for the freedom he gave me to develop my research. Balancing the research between classical and quantum was not an easy task, and his support made it possible to explore both sides throughout this work.

I am also grateful to my co-supervisors, Professor Anton Frisk Kockum and Professor Mats Granath, for their feedback, advice, and contributions. Their help was invaluable in allowing me to understand a little more of the quantum world and to navigate a field that was largely new to me.

I would also like to acknowledge Associate Professor Alexandru Paler, the discussion leader, for his comments and feedback on this thesis.

My thanks also go to everyone in my research group, past and present, for the insightful discussions as well as the many friendly moments we shared: André, Fareed, Luigi, Mateo, Mo, Stavroula, and Xu.

I also want to thank all the colleagues with whom I shared many lunches, fika, and barbecues. As a Swedish saying goes, *Ingen nämnd, ingen glömd*, meaning “no one mentioned, no one forgotten.” I will make one exception, however, for Lucia, since she and Luigi are the two colleagues and friends with whom I shared the most coffee breaks, and who have always been there to support me, cheer me on, or simply share a laugh that helped us get through the day.

Finally, I would like to thank my friends and family for their support and encouragement throughout these years. Grazie mille a tutti per avermi supportato e sopportato in questi anni impegnativi.

This work would not have been possible without the support of the Swedish Foundation for Strategic Research, which funded it through the SSF-FuSS QuantumStack project (grant number FUS21-0063).

Contents

Abstract	iii
List of Publications	v
Acknowledgement	vii

I Thesis Overview	1
1 Introduction	3
2 Quantum Computer Stack	4
3 Classical simulation of quantum computers	7
3.1 Simulations taxonomy	7
3.2 Algorithmic-level simulations	7
3.3 Simulation Hardware	8
4 From physical qubits to fault-tolerant computation	9
4.1 Quantum error correction (QEC) codes	10
4.2 Noise Models	10
4.3 Decoding Algorithms	11
4.4 GNN decoder	11
4.5 Decoding latency constraints	11
5 Research Questions	12
5.1 Simulations Acceleration	12
5.2 Real-time decoding	12
6 Thesis Contributions	13
6.1 Simulation Acceleration	13
6.2 Real-time decoding	13
7 Conclusions and Future Work	15

Bibliography	17
---------------------	-----------

II Appended Papers	21
---------------------------	-----------

Paper I - Simulation of quantum computers: Review and acceleration opportunities.

Paper II - Low Latency GNN Accelerator for Quantum Error Correction.

Part I

Thesis Overview

1 Introduction

Quantum computing is a promising emerging computing paradigm that exploits quantum-mechanical effects to process information. Its fundamental computational element is the quantum bit, or qubit. Whereas a classical bit takes one of the two values 0 or 1, a qubit can be in a linear combination of the two basis states:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle, \quad |\alpha|^2 + |\beta|^2 = 1, \quad (1)$$

where α and β are probability amplitudes. This property, known as superposition, allows a qubit to represent a combination of 0 and 1, while entanglement introduces correlations between qubits such that the state of the system must be described jointly rather than as a set of independent qubit states. When a qubit is measured in a basis called the computational basis, its state collapses to either 0 or 1, with probabilities determined by the corresponding probability amplitudes. Quantum computation is expressed as a circuit, where quantum gates act on one or more qubits to transform their state, and measurements convert the final quantum state into classical output [1].

By combining superposition, entanglement, interference, and measurement, quantum algorithms may solve selected classes of problems more efficiently than known classical approaches. Representative examples include Shor’s algorithm for integer factorization [2], and Grover’s algorithm for unstructured search [3]. The latter provides a quadratic reduction in the number of required search operations (Figure 1). Quantum algorithms are also being investigated for modelling naturally occurring quantum-mechanical systems, including applications in quantum chemistry [4], the development of new chemical compounds [5], and the evaluation of material properties [6]. Other approaches [7] target selected optimization problems (including variational quantum algorithms). More broadly, quantum speed-up is expected for specific problem classes where suitable quantum algorithms can exploit the structure of the computation.

The development of quantum computing remains in its early stages. Current machines have limited numbers of usable qubits, while noise, imperfect operations, measurement errors, and short qubit lifetimes restrict the depth of the quantum circuits that can be executed reliably. The size of an executable circuit is constrained both by the number of available qubits and by the number of operations that can be completed before accumulated errors make the result unreliable [1], [8]. These limitations currently prevent many promising quantum algorithms from being executed at the scale and fidelity required for practical advantage [8], [9]. A quantum computer therefore depends on substantial classical-computing support.

First, this thesis studies how to improve the scalability of simulation of quantum computers, connecting the bottlenecks of different simulation methods with acceleration opportunities across CPUs, GPUs, FPGAs, and distributed systems. Second, it studies how specialized classical hardware can support reliable quantum operation, developing an algorithm-hardware co-designed FPGA accelerator that enables a high-accuracy GNN surface-code decoder to meet real-time quantum error-correction requirements. Before examining classical

simulation and quantum error correction, the following section introduces the main components of a quantum computing system and briefly describes the role of each.

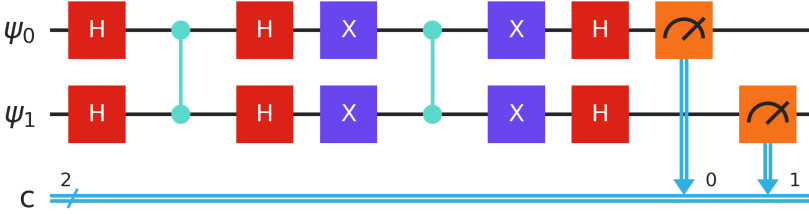


Figure 1: Example of a two-qubit quantum circuit implementing Grover’s search algorithm. The horizontal lines represent qubits, the boxes and connected symbols represent quantum gates applied over time from left to right, and the final measurements read the qubit states and store them in classical bits [10].

2 Quantum Computer Stack

A quantum computer is a heterogeneous system that combines the quantum chip with several classical components that support its operation. The classical system prepares the quantum program, translates it into hardware-supported operations, generates the signals that control the qubits, and processes the resulting measurements [9], [11]. Communication proceeds in two main directions. Along the control path, commands move from the classical host computer, which manages the quantum program and its execution, through the control electronics toward the quantum chip. Along the measurement path, analog signals produced by the chip are converted into digital values and returned to the classical system. Figure 2 presents a simplified view of this organization. In this figure, the controller represents several components that may be implemented separately, including pulse-generation hardware, data converters, readout logic, and classical feedback processing.

Quantum chips can be implemented using several technologies [12], [13], [14], [15], [16], with superconducting circuits and neutral atoms being among the most widely adopted. Although the implementation of individual components varies across technologies, their high-level system organization is broadly similar. This thesis focuses on superconducting quantum computers.

Superconducting quantum computers typically have constrained qubit connectivity, so a circuit may require additional routing operations, such as SWAP gates, when two interacting circuit qubits are mapped to physical qubits that cannot interact directly. The compiler may also consider differences in physical-qubit error rates, gate fidelities, readout errors, and calibration data when selecting a mapping. Finally, the compiler schedules the resulting operations

while respecting circuit dependencies and hardware timing constraints. The quality of this compilation directly affects the circuit depth, the number of additional operations, and the reliability of the final result.

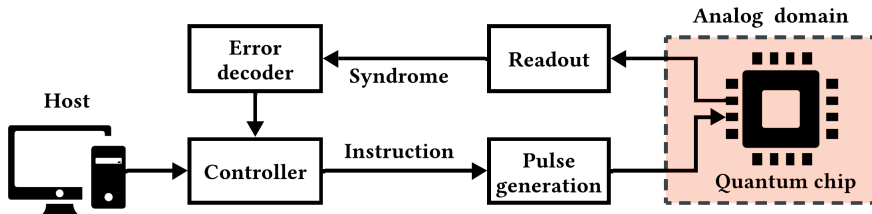


Figure 2: The host computer sends a quantum program to the controller, which drives the quantum chip with precise control pulses. The outputs of the chip are measured, and the error syndromes are computed from the observed errors. The controller may apply corrections based on those results [17].

The host computer provides the quantum program and manages its execution by initializing experiments, configuring their parameters, submitting circuits for execution, coordinating repeated executions, and collecting the resulting measurements. Before the program can be executed, a compiler translates the quantum algorithm into a circuit supported by the target hardware. This process includes decomposing operations into the native gate set and mapping the circuit qubits to the available physical qubits [9], [18]. Unlike an abstract circuit model, the compiler must account for hardware characteristics such as the native gate set, qubit connectivity, operation fidelities, and timing constraints.

The compiled operations are sent to the quantum controller, which translates them into commands for the pulse-generation hardware. In superconducting systems, qubits are controlled using radio-frequency signals rather than the digital voltage levels used by conventional logic. Each gate corresponds to one or more carefully configured control pulses. Pulse-generation hardware determines properties such as the pulse shape, frequency, amplitude, phase, and timing, according to the target operation and the current calibration of the device. Precise timing is necessary both to execute gates in the intended order and to coordinate operations that can run in parallel. The control pulses are initially represented digitally, for example as waveform samples or parameters used to generate a waveform. A Digital-to-Analog Converter (DAC) converts this digital representation into the analog signals applied to the quantum chip. Pulse generation and digital-to-analog conversion therefore form the interface between the discrete operations produced by the compiler and the physical signals used to manipulate the qubits.

The quantum chip contains the physical qubits together with the connections or couplers that allow them to interact. Single-qubit and multi-qubit gates are executed by applying the corresponding control pulses to the qubits and couplers. The native gate set and the physical connectivity of the chip determine which operations can be executed directly and which require additional compilation

and routing. These architectural constraints make the organization of the chip visible to the compiler, even when the original quantum algorithm is expressed independently of a particular device.

The duration and reliability of an execution are also limited by physical properties of the qubits. In particular, coherence time determines how long quantum information can be preserved [1], [19], while gate and measurement fidelities describe the reliability of the operations [20], [21]. A circuit must therefore complete before decoherence and accumulated operation errors make its output unreliable. Superconducting chips are typically operated at temperatures close to absolute zero, introducing further system constraints on the wiring and communication between the chip and the classical electronics.

The state of the qubits is measured at selected points during the circuit or at the end of its execution. Measurement produces analog signals that must be converted into classical information. These signals are first acquired through the measurement electronics and sampled using an Analog-to-Digital Converter (ADC). The resulting digital values are then processed by the readout system, which discriminates between the possible measurement outcomes, typically corresponding to the computational-basis states 0 and 1 defined in Equation 1. This classification is non-trivial because the measured signals are affected by noise, imperfect amplification, calibration drift, and overlap between the distributions associated with different states [21], [22]. The readout system therefore performs signal processing and classification before forwarding the measurement outcomes to the host or to other classical components. The ADC and readout logic form the return path from the analog behavior of the quantum chip to the digital representation used by the classical system.

In the case of quantum-error-corrected circuits, syndrome measurements are generated continuously, requiring a low-latency feedback path in which the controller processes measurement results at a rate that keeps pace with syndrome generation and uses them to determine subsequent actions. Instead of directly measuring the encoded quantum information, a quantum error correction (QEC) protocol repeatedly performs auxiliary measurements that reveal information about possible physical errors. For the surface-code approach considered in this thesis, changes in these measurements form error syndromes, which are processed by a decoder. The decoder identifies a likely set of errors and determines the corresponding correction or Pauli-frame update. QEC decoding is therefore part of the operational feedback loop rather than an offline analysis step. Its codes, decoding algorithms, and timing requirements are introduced in detail later in this thesis.

Within this stack, classical computation consequently supports quantum computing in several roles across the quantum stack. This thesis focuses on two of them.

The first is development, evaluation, and testing through classical simulation. Simulators reproduce the behavior of quantum devices, gates, or complete circuits without requiring execution on the physical machine. At the algorithmic level, computation and memory requirements can grow exponentially with the number of simulated qubits, depending on the simulation method and circuit structure, while memory bandwidth and data movement become major

architectural bottlenecks.

The second is classical processing during quantum-computer operation, particularly through quantum error correction. In this setting, the classical system processes a continuous stream of measurements and must return decoding decisions under strict latency or throughput constraints.

Classical simulation is therefore primarily a scalability-intensive workload, whereas QEC decoding additionally introduces real-time execution requirements. The following section examines the first of these roles by introducing classical quantum-computer simulation and the architectural challenges that limit its scalability.

3 Classical simulation of quantum computers

Classical simulation is an important tool for developing and evaluating quantum algorithms and quantum-computing systems. It supports algorithm development, testing and validation, compiler development, circuit analysis, and hardware and design-space exploration [8]. This role is particularly important while physical quantum computers remain limited in scale, reliability, and availability. Simulators also provide repeatability and visibility that are difficult to obtain from physical hardware: the same circuit can be executed under controlled conditions, intermediate states can be inspected, and alternative noise models, qubit mappings, or hardware assumptions can be compared. Simulation will therefore remain useful as quantum hardware improves, particularly for debugging, validation, and the exploration of algorithms and architectures before deployment.

3.1 Simulations taxonomy

The appropriate simulation method depends on the component being studied and on the required physical detail. Paper I organizes quantum-computer simulation into three abstraction levels: device, gate, and algorithmic level [8].

Device-level simulation models materials, geometry, and physical implementation of qubits, including electromagnetic, thermal, and mechanical effects.

Gate-level simulation focuses on a small number of qubits and couplers, often modelling their time evolution under control pulses to study energy levels, interactions, and pulse parameters.

Algorithmic-level simulation treats quantum gates as mathematical transformations and evaluates complete quantum circuits. Moving to a higher abstraction level removes physical detail but allows larger systems to be studied. This thesis focuses on algorithmic-level simulation, where computation, memory, and data movement become central computer-architecture challenges.

3.2 Algorithmic-level simulations

The most direct algorithmic-level method is Schrödinger-style, or state-vector, simulation [8], [23]. An n -qubit state is represented by 2^n complex probability amplitudes, and each gate updates the relevant groups of amplitudes. The

method is general and conceptually simple, but its memory requirement grows exponentially with the number of qubits: adding one qubit doubles the state-vector size. Gate execution also requires repeated accesses to this increasingly large vector. Performance can therefore be limited by memory capacity, memory bandwidth, and locality in addition to arithmetic throughput. When the state is distributed across several processors or accelerators, communication among their separate memory spaces introduces an additional cost [24].

Feynman-style path summation [8], [25] instead evaluates the contributions of computational paths through the circuit and sums them to obtain the final result. Individual paths require comparatively little memory and can be processed independently, but their number may grow exponentially with branching operations. Feynman-style methods trade storage for additional computation. Schrödinger-Feynman hybrid methods [8] partition the qubits or circuit, use state-vector simulation within each partition, and evaluate gates crossing partition boundaries through path-based decomposition. Their performance depends strongly on the selected partition and on the number of cross-partition gates, allowing memory consumption to be reduced at the cost of additional execution time.

Unlike the state-vector and path-based approaches described above, tensor-based simulation represents the quantum circuit as a tensor network and evaluates it through a sequence of tensor contractions. Tensor-based simulation [8], [26] represents states and gates as a tensor network and evaluates the circuit through a sequence of tensor contractions. It does not necessarily construct the complete 2^n -element state and can therefore be effective for circuits with many qubits but favorable connectivity or depth, or when only selected output amplitudes are required. Its cost depends on the contraction order and on the largest intermediate tensor created during the process. Slicing limits intermediate sizes by fixing selected indices and evaluating several smaller contractions. This reduces memory pressure and creates independent tasks for parallel execution, but it may also repeat computation.

The methods consequently expose different architectural bottlenecks. State-vector simulation performs regular numerical operations over exponentially growing data, making memory capacity, bandwidth, and data movement major constraints. Noisy simulations may require more expensive representations, such as density matrices, further increasing computation and storage. Distributed execution adds communication and synchronization among nodes or devices. Tensor-based performance additionally depends on contraction-path selection, intermediate tensor size, index permutation, slicing, and load balancing. The best method depends on the circuit structure, the requested output, the numerical representation, and the available hardware.

3.3 Simulation Hardware

CPU acceleration focuses on locality, parallel execution, and distributed scalability. Circuit partitioning divides the workload into tasks, while gate clustering combines nearby gates so that several operations can be performed during one traversal of the state vector. Cache blocking and gate reordering improve

locality, and multithreading, SIMD execution, and distributed processing expose parallelism at different levels [23], [27]. Tensor-based implementations additionally optimize tensor-network construction, contraction order, slicing, and task distribution. Compression and reduced data formats can lower memory requirements, although their overhead and effect on accuracy must be considered.

GPUs provide high parallelism for independent amplitude updates and tensor contractions. Gate fusion [28] increases the work performed for each access to the state vector, while shared-memory blocking, qubit reordering, and optimized data layouts reduce global-memory traffic. Efficient implementations also minimize transfers between the CPU and GPU. Dense parts of the computation can be mapped to optimized matrix and tensor libraries and to specialized matrix-processing units. Multi-GPU execution extends the available computation and memory capacity, but communication between devices becomes increasingly important.

FPGAs provide fewer general-purpose resources than high-end CPUs and GPUs, but allow the datapath and numerical representation to be specialized. Arbitrary-precision arithmetic can reduce storage and resource costs, lookup tables can replace expensive calculations, and custom complex multiply-and-accumulate units can match the dataflow of gate operations. FPGA designs can also avoid operations involving zero or otherwise simple matrix elements and stream data through pipelined arithmetic units.

Across all platforms, recurring opportunities include reduced or mixed precision, data compression, parallel execution, memory-locality optimization, and dedicated matrix-processing hardware. These techniques address different computation-memory bottlenecks and must be evaluated together with their effects on accuracy, flexibility, and scalability.

4 From physical qubits to fault-tolerant computation

Physical qubits are affected by decoherence, noise from interactions with the environment, imperfect control pulses, gate errors, and measurement errors. Consequently, the probability that a quantum circuit executes correctly decreases as the circuit depth increases. Current superconducting qubits typically have physical error rates on the order of 10^{-3} , whereas useful quantum algorithms may require millions or billions of reliable quantum operations [29]. Improvements at the device level alone are therefore insufficient, motivating Quantum Error Correction (QEC) to reduce the effective error rate and enable fault-tolerant quantum computation [1], [30].

QEC cannot directly apply the redundancy mechanisms used in classical systems because an unknown quantum state cannot be freely measured or copied: measurement generally disturbs the state, while the no-cloning principle prevents arbitrary quantum states from being duplicated [1].

QEC encodes one logical qubit across multiple physical qubits rather than copying the quantum state directly. Data qubits store the encoded information,

while measurement qubits are used to measure stabilizers, which are parity-like constraints involving neighboring data qubits [1]. Repeated stabilizer measurements reveal changes caused by physical faults without directly measuring the encoded logical state. A decoder processes these syndromes and determines a correction or Pauli-frame update. A decoding failure occurs when the combined physical error and correction result in a logical error.

The main application-level metric for a decoder is its logical error rate, defined as

$$p_L = \frac{\text{number of decoding instances with a logical failure}}{\text{total number of decoding instances}}. \quad (2)$$

For fault-tolerant execution, this rate must be substantially lower than the physical error rate. QEC achieves this reduction at the cost of additional physical qubits, measurements, and classical processing.

4.1 Quantum error correction (QEC) codes

Different QEC codes use different strategies to encode and protect logical information. Among them, the surface code [31] is one of the standard approaches for superconducting quantum computers, as it is compatible with two-dimensional qubit layouts and relies primarily on local interactions between neighboring qubits. The surface code also provides a comparatively high error threshold: below this threshold, increasing the size of the code reduces the logical error rate [31], [32].

The size and error-correction capability of the surface code are characterized by its code distance d . A logical qubit of distance d uses d^2 data qubits and $d^2 - 1$ measurement qubits and can correct up to $\lfloor (d-1)/2 \rfloor$ single-qubit errors. Increasing the code distance improves error protection, but also increases the number of physical qubits and syndrome measurements that must be processed by the classical system.

Since the syndrome-extraction process is itself affected by circuit-level noise [31], [32], measurements must be repeated over time. A common approach is to perform $d_t = d$ measurement rounds, allowing errors in the data qubits, measurement qubits, and readout circuitry to be detected. Changes between consecutive stabilizer measurements produce detection events, which can be represented as nodes in a space-time syndrome graph. Consequently, increasing the code distance increases both the number of measurements and the potential size of this graph, resulting in a larger classical decoding workload.

4.2 Noise Models

Different noise models are used to evaluate quantum error-correction codes, trading physical realism against simulation complexity [32], [33].

In the code-capacity noise model, errors are applied only to the data qubits, while syndrome measurements are assumed to be perfect.

The phenomenological noise model additionally allows errors in the syndrome measurements, requiring repeated measurement rounds to distinguish data-qubit errors from measurement errors.

Circuit-level noise provides a more detailed model in which faults can occur during the individual operations of the syndrome-extraction circuit, including single- and multi-qubit gates, state preparation, measurement, and idle operations.

As the noise model becomes more detailed, it more closely represents the behavior of a physical implementation but also increases the complexity of simulation and decoding. Paper II uses circuit-level noise, consistent with the GNN decoder on which the work is based.

4.3 Decoding Algorithms

Several algorithms have been proposed to decode surface-code syndromes, offering different trade-offs between accuracy and computational complexity. Union-Find [34] is a relatively lightweight approach, but generally achieves a higher logical error rate. Minimum-Weight Perfect Matching [35] formulates decoding as a weighted matching problem and is widely used because of its strong balance between decoding accuracy and implementation cost, with several hardware-accelerated implementations targeting real-time operation. Belief-propagation [36], [37] methods provide another alternative and are particularly relevant for quantum low-density parity-check codes. More recently, neural-network-based decoders have been explored, offering a broader design space in which decoding accuracy and computational cost depend on the selected model architecture and training method. Graph Neural Networks (GNNs) [33], [38] are particularly well suited to this setting because quantum error-correction problems can be represented using graph-structured data, allowing message passing to capture correlations relevant to decoding.

4.4 GNN decoder

The GNN selected in Paper II represents the syndrome as a variable-size graph [38]. Nodes correspond to detection events and encode their stabilizer type and space-time position, while edges connect nearby events. Graph-convolution layers perform message passing by updating each node representation using information from the node itself and its neighbors. The resulting node representations are then combined using global mean pooling, which averages the features across all nodes to obtain a fixed-size representation of the entire graph. Finally, fully connected, or dense, layers process this graph-level representation to produce the logical Pauli-frame update [38], a classical record of the correction inferred by the decoder that can be tracked rather than physically applied immediately [32].

4.5 Decoding latency constraints

Real-time decoding [30] is required because syndrome measurements are generated continuously during quantum execution. If decoding is slower than syndrome generation, measurements accumulate and the classical control system could become a bottleneck [32], [39]. For superconducting surface codes, a

syndrome cycle may be as short as approximately $1\text{ }\mu\text{s}$ [40]. Two timing models are commonly considered. The first corresponds to a hard real-time constraint: in *max-latency* decoding, every supported decoding instance must complete within one syndrome cycle, imposing a strict per-instance deadline. The second is closer to a soft real-time or throughput constraint: in *average-latency* decoding, individual instances may take longer, provided that the average decoding rate remains faster than the syndrome-generation rate. The latter can exploit buffering or parallel decoding and allows more variation in execution time.

5 Research Questions

5.1 Simulations Acceleration

The performance of a quantum-computer simulator depends jointly on the simulation method, the circuit, and the target classical architecture. Existing acceleration work spans different abstraction levels, circuit classes, platforms, and performance metrics. Some studies prioritize the maximum number of simulated qubits, while others focus on execution time, memory consumption, fidelity, or distributed scalability. Reported speedups alone therefore do not explain which bottleneck is addressed, which hardware feature enables the improvement, or which trade-off is introduced. This motivates the following research question:

What are the main computational and memory bottlenecks of quantum-computer simulation, and which hardware-aware techniques can improve its performance and scalability on CPUs, GPUs, and FPGAs?

5.2 Real-time decoding

The selected GNN [38] achieves a lower logical error rate than MWPM, but its computational cost prevents direct real-time execution. The original model [38] contains approximately 1.4×10^6 parameters, and the cost of its graph-convolution layers increases with the number of input nodes. In software, inference requires approximately $291.4\text{ }\mu\text{s}$ per graph, far above the sub-microsecond target required for superconducting chips. Meeting the real-time constraint therefore requires reducing the computational and storage requirements of the decoder while preserving its accuracy advantage. The research question addressed by Paper II is:

How can high-accuracy GNN-based quantum error-correction decoders be optimized for specialized hardware to meet real-time decoding requirements while preserving decoding accuracy?

6 Thesis Contributions

6.1 Simulation Acceleration

Paper I, **Simulation of Quantum Computers: Review and Acceleration Opportunities**, presents a survey of quantum-computer simulation with a particular focus on acceleration using classical hardware [8]. Rather than proposing a new simulator, the paper provides a structured view of the simulation landscape and analyzes existing approaches from a computer-architecture perspective.

The first contribution is a classification of quantum-computer simulation according to the abstraction level at which the system is modeled. This organization connects device-, gate-, and algorithmic-level simulation and places the different simulation tools and techniques within a common framework. The paper then focuses on algorithmic-level simulation, where the computation and memory requirements make hardware acceleration particularly relevant.

The second contribution of the survey is its hardware-aware analysis of simulation acceleration. Existing approaches are organized according to the classical platform used for execution, including CPUs, GPUs, FPGAs, and distributed or heterogeneous systems. Rather than considering the reported performance of individual simulators in isolation, the survey relates the proposed optimizations to the architectural bottlenecks they address, such as computation, memory capacity, memory bandwidth, data movement, and communication. This provides a common perspective for comparing techniques developed for different simulation methods and hardware platforms.

Finally, Paper I identifies recurring hardware-aware optimization opportunities across the reviewed literature. These include reduced and mixed precision [23], [41], data compression [42], parallel execution [23], [24], improved memory locality [23], [27], and the use of specialized hardware for matrix operations [28].

The paper discusses possible directions for improving simulation performance and scalability, including exploiting hardware originally developed for machine-learning workloads and further specialization using reconfigurable architectures. Overall, the contributions of Paper I are the organization and synthesis of existing quantum-computer simulation research from a hardware-acceleration perspective, providing a useful starting point for identifying promising directions for further research.

6.2 Real-time decoding

Paper II, **Low Latency GNN Accelerator for Quantum Error Correction**, develops a real-time FPGA accelerator for the selected GNN [38] surface-code decoder through a sequence of algorithm- and architecture-level optimizations [17]. The GNN is organized as a sequence of computational layers, where each layer transforms the representation produced by the previous one. Its graph-convolution layers account for most of the computational cost because they process every node in the input graph, while the later pooling and dense layers operate on the resulting graph-level representation.

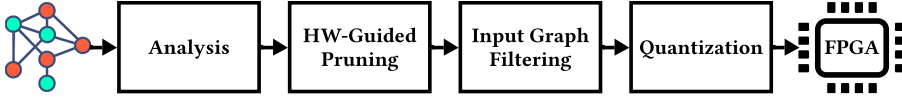


Figure 3: Overview of the optimizations used to map the GNN decoder to FPGA.

Rather than mapping the original model directly to hardware, the work first characterizes its computational bottlenecks, derives separate models for the two timing requirements, bounds the input-dependent workload, and selects fixed-point representations, as shown in Figure 3. The resulting models are then mapped to a custom FPGA architecture and hardware execution schedule.

The first contribution is a layer-level analysis of the original GNN, considering parameter count, multiplication count, activation sparsity, storage, and dependence on input size. Graph-convolution layers dominate the workload because their operations are repeated for every graph node, with the largest layer requiring $524,288N$ multiplications. With $d_t = d$, the theoretical maximum graph size is

$$N_{\max} = (d^2 - 1)d_t. \quad (3)$$

For $d = 7$, $N_{\max} = 336$, resulting in roughly 10^8 multiplications for the original network, which is incompatible with the one-microsecond target. Profiling also shows substantial and non-uniform activation sparsity across **GraphConv** layers. Paper II defines a hardware-guided pruning order based on the number of avoidable multiplications, prioritizing layers that are both computationally expensive and highly sparse.

The second contribution is hardware-guided pruning and retraining. Direct layer removal significantly degrades decoding accuracy. Paper II instead prunes selected layers and retrains the remaining network, producing two configurations with different latency-accuracy trade-offs. The max-latency model removes **GraphConv3-6** and computes only half of the **GraphConv2** outputs. It contains approximately 1.5×10^5 parameters, a $9.2\times$ reduction from the original model. The average-latency model removes **GraphConv3-5** and retains approximately 3.1×10^5 parameters, corresponding to a $4.4\times$ reduction. The two models trade more aggressive complexity reduction for stricter latency requirements against higher decoding accuracy.

The third contribution is latency-bounded input-graph filtering. Worst-case syndrome graphs are much larger than those typically observed at the evaluated physical error rate. Provisioning the accelerator for $N_{\max} = 336$ would therefore consume substantial resources for rare inputs. Paper II instead bounds the supported graph size using the measured graph-size distribution. For $d = 7$, the max-latency decoder supports up to $N = 30$ nodes, while the average-latency decoder supports up to $N = 32$. Larger graphs are discarded and assigned a default no-error output, with the resulting failures included in the reported logical error rate. Bounding the graph size reduces storage, limits the number of **GraphConv** iterations, and places an upper bound on execution time. It is particularly important for max-latency decoding, where every supported input

must meet the deadline.

The fourth contribution is post-training quantization. The pruned models are further reduced through post-training quantization of weights, features, biases, and accumulators. The selected formats are chosen by considering logical error rate, latency, and FPGA resource utilization together. The max-latency model uses 14-bit weights, 17-bit features, 5-bit biases, and 27-bit accumulation, while the average-latency model uses slightly wider feature and accumulator formats. These precisions allow each multiplication to map to a single DSP, increasing parallelism and reducing storage and arithmetic costs.

The last contribution is the design of the FPGA-based architecture. The optimized models are mapped to a parameterized three-stage FPGA architecture. The first stage prepares node inputs and neighbor aggregation, the second performs DSP-based multiplications, and the third completes accumulation, bias addition, and output generation. Large layers are folded across multiple cycles when the available DSP parallelism is insufficient. Additional optimizations include interleaving neighbor aggregation with computation, distributing packed weights across BRAM banks, reusing compatible adder trees, and replacing global mean-pooling division with precomputed normalization factors. These choices reduce latency and resource utilization while matching the pruned network structure.

Evaluations are performed on surface codes up to $d = 7$ with circuit-level noise at $p = 10^{-3}$ using 10^8 simulated input graphs. The FPGA design is synthesized for a Xilinx Alveo U250, with latency measured from receipt of the constructed graph to decoder output. For max-latency decoding, the final design completes the largest supported input in 988.8 ns and achieves a logical error rate of 1.47×10^{-5} , 13% lower than the evaluated MWPM baseline [43]. For average-latency decoding, the larger model achieves an average latency of 846 ns and a logical error rate of 1.01×10^{-5} , corresponding to a 40% improvement over the evaluated MWPM baseline [43].

These results show that the GNN accuracy advantage can be preserved while meeting both real-time constraints. More generally, they highlight the need to co-design model complexity, numerical representation, input size, hardware architecture, and execution schedule.

7 Conclusions and Future Work

This thesis has explored selected classical-computing challenges arising in the development and reliable operation of quantum computers from a computer-architecture perspective. On the development side, it has considered the computational and memory challenges of simulating quantum computers on classical hardware and the opportunities for hardware-aware acceleration. On the operational side, it has considered the real-time classical processing required for quantum error correction and how specialized hardware can help satisfy its demanding latency and resource constraints. Together, these two directions illustrate the importance of the classical computing system surrounding the quantum processor and the opportunities for computer-architecture techniques

to improve its performance and scalability.

There are several promising directions for future work on the classical side of quantum computing. One direction is to move beyond the acceleration of individual quantum error-correction components and consider the scheduling of complete QEC workloads. As fault-tolerant systems scale, different quantum and classical operations must be coordinated while satisfying dependencies, resource constraints, and strict timing requirements. Investigating how these operations should be scheduled, and how the available classical processing resources should be allocated, is therefore an important step toward scalable quantum error-corrected systems.

A broader direction is the architecture of the quantum control system itself. Quantum execution requires continuous interaction between the quantum processor and classical components responsible for control, measurement, feedback, and error correction. As the number of qubits and the amount of real-time processing grow, the organization of this classical system and the communication between its components become increasingly important.

Future work will therefore investigate how computer-architecture techniques can support the control of larger quantum computers and how scheduling, computation, and communication can be jointly considered when designing these systems.

The remainder of the thesis contains the two publications on which this work is based. Paper I, **Simulation of Quantum Computers: Review and Acceleration Opportunities**, reviews quantum-computer simulation and hardware-aware acceleration techniques for classical platforms. Paper II, **Low Latency GNN Accelerator for Quantum Error Correction**, presents an algorithm-hardware co-designed FPGA accelerator for real-time surface-code decoding. The two papers are reproduced in the following chapters and are referred to throughout the thesis as Paper I and Paper II.

Bibliography

- [1] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*. Cambridge University Press, 2023 (cit. on pp. 3, 6, 9, 10).
- [2] P. Shor, ‘Algorithms for quantum computation: discrete logarithms and factoring,’ in *Proceedings 35th Annual Symposium on Foundations of Computer Science*, ser. SFCS ’94, Washington, DC, USA: IEEE Comput. Soc. Press, 1994, p. 124. DOI: 10.1109/SFCS.1994.365700 (cit. on p. 3).
- [3] L. K. Grover, ‘Quantum mechanics helps in searching for a needle in a haystack,’ *Physical review letters*, vol. 79, no. 2, p. 325, 1997. DOI: 10.1103/PhysRevLett.79.325 (cit. on p. 3).
- [4] S. McArdle, S. Endo, A. Aspuru-Guzik, S. C. Benjamin and X. Yuan, ‘Quantum computational chemistry,’ *Reviews of Modern Physics*, vol. 92, p. 015003, 1 2020. DOI: 10.1103/RevModPhys.92.015003 [Online]. Available: <https://link.aps.org/doi/10.1103/RevModPhys.92.015003> (cit. on p. 3).
- [5] A. M. Dalzell et al., *Quantum algorithms: A survey of applications and end-to-end complexities*, 2023. arXiv: 2310.03011 (cit. on p. 3).
- [6] B. Bauer, S. Bravyi, M. Motta and G. K.-L. Chan, ‘Quantum algorithms for quantum chemistry and quantum materials science,’ *Chemical Reviews*, vol. 120, no. 22, pp. 12 685–12 717, 2020 (cit. on p. 3).
- [7] R. Shaydulin, H. Ushijima-Mwesigwa, C. F. A. Negre, I. Safro, S. M. Mniszewski and Y. Alexeev, ‘A hybrid approach for solving optimization problems on small quantum computers,’ *Computer*, vol. 52, no. 6, pp. 18–26, 2019. DOI: 10.1109/MC.2019.2908942 (cit. on p. 3).
- [8] A. Cicero, M. A. Maleki, M. W. Azhar, A. F. Kockum and P. Trancoso, *Simulation of quantum computers: Review and acceleration opportunities*, 2025. arXiv: 2410.12660 [quant-ph] (cit. on pp. 3, 7, 8, 13).
- [9] M. Bandic, S. Feld and C. G. Almudever, ‘Full-stack quantum computing systems in the NISQ era: Algorithm-driven and hardware-aware compilation techniques,’ in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 2022, pp. 1–6 (cit. on pp. 3–5).

- [10] Gadi Aleksandrowicz, et al., *Qiskit: An open-source framework for quantum computing*, Zenodo, 2019. DOI: 10.5281/zenodo.2573505 (cit. on p. 4).
- [11] Y. Xu et al., ‘QubiC: An open-source FPGA-based control and measurement system for superconducting quantum information processors,’ *IEEE Transactions on Quantum Engineering*, vol. 2, pp. 1–11, 2021 (cit. on p. 4).
- [12] D. Kielpinski, C. Monroe and D. Wineland, ‘Architecture for a large-scale ion-trap quantum computer,’ *Nature*, vol. 417, pp. 709–11, Jul. 2002. DOI: 10.1038/nature00784 (cit. on p. 4).
- [13] C. Young et al., ‘An architecture for quantum networking of neutral atom processors,’ *Applied Physics B*, vol. 128, Aug. 2022. DOI: 10.1007/s00340-022-07865-0 (cit. on p. 4).
- [14] F. P. G. de Arquer, D. V. Talapin, V. I. Klimov, Y. Arakawa, M. Bayer and E. H. Sargent, ‘Semiconductor quantum dots: Technological progress and future challenges,’ *Science*, vol. 373, no. 6555, eaaz8541, 2021. DOI: 10.1126/science.aaz8541 eprint: <https://www.science.org/doi/pdf/10.1126/science.aaz8541>. [Online]. Available: <https://www.science.org/doi/abs/10.1126/science.aaz8541> (cit. on p. 4).
- [15] S. Omkar, S.-H. Lee, Y. S. Teo, S.-W. Lee and H. Jeong, ‘All-photonic architecture for scalable quantum computing with Greenberger-Horne-Zeilinger States,’ *PRX Quantum*, vol. 3, p. 030309, 3 2022. DOI: 10.1103/PRXQuantum.3.030309 [Online]. Available: <https://link.aps.org/doi/10.1103/PRXQuantum.3.030309> (cit. on p. 4).
- [16] J. M. Gambetta, J. M. Chow and M. Steffen, ‘Building logical qubits in a superconducting quantum computing system,’ *npj quantum information*, vol. 3, no. 1, p. 2, 2017 (cit. on p. 4).
- [17] A. Cicero, L. Altamura, M. Lange, M. Granath and P. Trancoso, *Low-latency GNN accelerator for quantum error correction*, Manuscript included in this thesis, 2026 (cit. on pp. 5, 13).
- [18] C. Zhang, A. B. Hayes, L. Qiu, Y. Jin, Y. Chen and E. Z. Zhang, ‘Time-optimal qubit mapping,’ in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS ’21, Virtual, USA: Association for Computing Machinery, 2021, 360–374, ISBN: 9781450383172. DOI: 10.1145/3445814.3446706 [Online]. Available: <https://doi.org/10.1145/3445814.3446706> (cit. on p. 5).
- [19] M. H. Devoret and R. J. Schoelkopf, ‘Superconducting circuits for quantum information: An outlook,’ *Science*, vol. 339, no. 6124, pp. 1169–1174, 2013. DOI: 10.1126/science.1231930 eprint: <https://www.science.org/doi/pdf/10.1126/science.1231930>. [Online]. Available: <https://www.science.org/doi/abs/10.1126/science.1231930> (cit. on p. 6).

- [20] S. Kosen et al., ‘Building blocks of a flip-chip integrated superconducting quantum processor,’ *Quantum Science and Technology*, vol. 7, no. 3, p. 035 018, 2022. DOI: 10.1088/2058-9565/ac734b [Online]. Available: <https://dx.doi.org/10.1088/2058-9565/ac734b> (cit. on p. 6).
- [21] L. Chen et al., ‘Transmon qubit readout fidelity at the threshold for quantum error correction without a quantum-limited amplifier,’ *npj Quantum Information*, vol. 9, no. 1, p. 26, 2023, ISSN: 2056-6387. DOI: 10.1038/s41534-023-00689-6 [Online]. Available: <https://doi.org/10.1038/s41534-023-00689-6> (cit. on p. 6).
- [22] M. Kjaergaard et al., ‘Superconducting qubits: Current state of play,’ *Annual Review of Condensed Matter Physics*, vol. 11, no. 1, pp. 369–395, 2020 (cit. on p. 6).
- [23] A. Fatima and I. L. Markov, ‘Faster schrödinger-style simulation of quantum circuits,’ in *Proceedings of the IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2021, pp. 194–207 (cit. on pp. 7, 9, 13).
- [24] A. Li et al., ‘Sv-sim: Scalable pgas-based state vector simulation of quantum circuits,’ in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021, pp. 1–14 (cit. on pp. 8, 13).
- [25] E. Bernstein and U. Vazirani, ‘Quantum complexity theory,’ *SIAM Journal on Computing*, vol. 26, no. 5, pp. 1411–1473, 1997. DOI: 10.1137/S0097539796300921 eprint: <https://doi.org/10.1137/S0097539796300921>. [Online]. Available: <https://doi.org/10.1137/S0097539796300921> (cit. on p. 8).
- [26] I. L. Markov and Y. Shi, ‘Simulating quantum computation by contracting tensor networks,’ *SIAM Journal on Computing*, vol. 38, no. 3, pp. 963–981, 2008. DOI: 10.1137/050644756 eprint: <https://doi.org/10.1137/050644756>. [Online]. Available: <https://doi.org/10.1137/050644756> (cit. on p. 8).
- [27] B. Fang, M. Y. Özkaya, A. Li, Ü. V. Çatalyürek and S. Krishnamoorthy, ‘Efficient hierarchical state vector simulation of quantum circuits via acyclic graph partitioning,’ in *2022 IEEE International Conference on Cluster Computing (CLUSTER)*, IEEE, 2022, pp. 289–300 (cit. on pp. 9, 13).
- [28] H. Bayraktar et al., ‘cuQuantum SDK: A high-performance library for accelerating quantum science,’ in *Proceedings of the IEEE International Conference on Quantum Computing and Engineering (QCE)*, vol. 1, 2023, pp. 1050–1061 (cit. on pp. 9, 13).
- [29] C. Gidney and M. Ekerå, ‘How to factor 2048 bit rsa integers in 8 hours using 20 million noisy qubits,’ *Quantum*, vol. 5, p. 433, 2021 (cit. on p. 9).

- [30] F. Battistel et al., ‘Real-time decoding for fault-tolerant quantum computing: Progress, challenges and outlook,’ *Nano Futures*, vol. 7, no. 3, p. 032 003, 2023 (cit. on pp. 9, 11).
- [31] A. G. Fowler, M. Mariantoni, J. M. Martinis and A. N. Cleland, ‘Surface codes: Towards practical large-scale quantum computation,’ *Physical Review A—Atomic, Molecular, and Optical Physics*, vol. 86, no. 3, p. 032 324, 2012 (cit. on p. 10).
- [32] B. M. Terhal, ‘Quantum error correction for quantum memories,’ *Reviews of Modern Physics*, vol. 87, no. 2, pp. 307–346, 2015 (cit. on pp. 10, 11).
- [33] A. S. Maan and A. Paler, ‘Machine learning message-passing for the scalable decoding of qldpc codes,’ *npj Quantum Information*, vol. 11, no. 1, p. 78, 2025 (cit. on pp. 10, 11).
- [34] N. Delfosse and N. H. Nickerson, ‘Almost-linear time decoding algorithm for topological codes,’ *Quantum*, vol. 5, p. 595, 2021 (cit. on p. 11).
- [35] V. Kolmogorov, ‘Blossom v: A new implementation of a minimum cost perfect matching algorithm,’ *Mathematical Programming Computation*, vol. 1, pp. 43–67, 2009 (cit. on p. 11).
- [36] J. Old and M. Rispler, ‘Generalized belief propagation algorithms for decoding of surface codes,’ *Quantum*, vol. 7, p. 1037, 2023 (cit. on p. 11).
- [37] N. P. Breuckmann and J. N. Eberhardt, ‘Quantum low-density parity-check codes,’ *Prx Quantum*, vol. 2, no. 4, p. 040 101, 2021 (cit. on p. 11).
- [38] M. Lange et al., ‘Data-driven decoding of quantum error correcting codes using graph neural networks,’ *Physical Review Research*, vol. 7, no. 2, p. 023 181, 2025 (cit. on pp. 11–13).
- [39] L. Skoric, D. E. Browne, K. M. Barnes, N. I. Gillespie and E. T. Campbell, ‘Parallel window decoding enables scalable fault-tolerant quantum computation,’ *Nature Communications*, vol. 14, p. 7040, 2023 (cit. on p. 11).
- [40] Google Quantum AI, ‘Suppressing quantum errors by scaling a surface code logical qubit,’ *Nature*, vol. 614, no. 7949, pp. 676–681, 2023 (cit. on p. 12).
- [41] Y. Liu et al., ‘Closing the” quantum supremacy” gap: achieving real-time simulation of a random quantum circuit using a new sunway supercomputer,’ in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021, pp. 1–12 (cit. on p. 13).
- [42] X.-C. Wu et al., ‘Full-state quantum circuit simulation by using data compression,’ in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2019 (cit. on p. 13).
- [43] O. Higgott, ‘Pymatching: A python package for decoding quantum codes with minimum-weight perfect matching,’ *ACM Transactions on Quantum Computing*, vol. 3, no. 3, pp. 1–16, 2022 (cit. on p. 15).