VILNIUS UNIVERSITY
FACULTY OF MATHEMATICS AND INFORMATICS
SOFTWARE ENGINEERING STUDY PROGRAM

Research of Quantum Computer Topologies for Optimization Algorithms

Kvantinių kompiuterių topologijų tyrimas optimizavimo algoritmams

Master thesis

Author: Gabrielius Keibas

Supervisor: doc. dr. Linas Petkevičius

Reviewer: Irus Grinis

Vilnius – 2026

Abstract

Efficient execution of quantum optimisation algorithms on near-term quantum hardware remains challenging due to limited qubit connectivity. This work presents QuMeld, an open-source framework for topology-aware optimisation of quantum circuits. It is used to systematically compare six qubit mapping algorithms across sixteen quantum computer topologies and six benchmark circuits covering VQE and QAOA workloads. Results reveal that quantum optimisation algorithms do not reduce the qubit mapping problem to a simplified form. Preliminary decision tree rules achieved 45,7 – 72,3 % accuracy, outperforming the random baseline (16,7 %). While this exceeds random selection, it is insufficient to replace exhaustive mapper evaluation in practice. The framework was validated on real IBM hardware with results that demonstrated end-to-end hardware compatibility, and a quantum machine learning experiment on a medical benchmark dataset showed practical end-to-end usability. Overall, no single mapper is globally optimal, including default framework mappers, and criterion-driven exhaustive evaluation across candidate mappers is the recommended practical strategy in the current NISQ era.

Keywords: quantum circuit mapping, qubit mapping problem, quantum optimization algorithms, quantum computer topologies

Santrauka

Kvantinių optimizavimo algoritmų vykdymas artimojo laikotarpio kvantiniuose kompiuteriuose išlieka sudėtinga problema dėl riboto kubitų jungumo. Šiame darbe yra pristatomas atviro kodo kvantinių schemų optimizavimo testavimo karkasas QuMeld. Naudojant šį karkasą, tyrimui buvo pasirinkti šeši schemų optimizavimo algoritmai, šešios kvantinės schemos, susijusios su VQE ir QAOA algoritmais, ir šešiolika kvantinių kompiuterių topologijų. Gauti tyrimo rezultatai rodo, kad kvantiniai optimizavimo algoritmai nesupaprastina kubitų atvaizdavimo problemos. Preliminarūs sprendimų medžiai pasiekė 45,7 – 72,3 % tikslumą, pranokdami atsitiktinį algoritmo pasirinkimą (16,7 %). Tokio tikslumo nepakanka, kad būtų galima atsisakyti pilno algoritmų perrinkimo praktikoje. Karkaso veikimas buvo patvirtintas ant tikro IBM kvantinio kompiuterio, o kvantinio mašininio mokymosi eksperimentas su medicininių duomenų rinkiniu parodė praktinį karkaso pritaikomumą. Apibendrinant, nė vienas schemų optimizavimo algoritmas nėra globaliai optimalus, įskaitant numatytuosius karkasų algoritmus, todėl NISQ eroje rekomenduojama perrinkti visus galimus algoritmus pagal pasirinktą optimizavimo kriterijų.

Raktiniai žodžiai: kvantinių schemų optimizavimas, kubitų atvaizdavimo problema, kvantiniai optimizavimo algoritmai, kvantinių kompiuterių topologijos

Contents

Introduction

Quantum Computing is a rapidly growing technology that has an impact on many different industries like biology [CCC+20], chemistry [CRO+19], manufacturing processes [PSK24], finances [HGL+23] and more. It uses principles of quantum mechanics to perform complex calculations in parallel. This allows for a huge speed up in a subset of algorithms compared to a traditional computer. These algorithms are run on quantum computers. They differ in their technology: superconducting [WBC+21], photonic [PMS+14], trapped ions [MK13], neutral atoms [WDE+23]. Due to their differences in physical manner, they also differ in the topologies. Quantum computer topology is a graph that represents how the qubits are connected with each other. Qubit is how the data is stored inside a quantum computer. It is a quantum equivalent of the traditional bit used by the classical computer. At the time of writing, the most accessible quantum computers have around 105–156 qubits. This is a sufficient amount of qubits to see a potential of these computers. However, the biggest impact for the current processes will be when quantum computers will have thousands, tens thousands of qubits. Currently, this is not achievable due to the errors. Errors are happening in the qubits and also on the edges between them. Each qubit is connected to a subset of qubits, according to the topology. This is the reason why the current quantum era is called Noisy intermediate-scale quantum era (NISQ) [Pre18].

Majority of algorithms are using two or more qubits gates. To apply these gates on the physical quantum computers can sometimes be impossible due to the topology. To solve this issue dynamic remapping of logical qubits to physical qubits in the compiler is needed to enable these gates in the desired algorithm. Dedicated algorithms have to be used that would modify quantum circuit (representation of a quantum algorithm – a sequence of quantum gates, measurements and qubits) in that manner that it would produce the same result and would be runnable on the selected quantum computer. This problem is called qubit mapping problem.

One of the desired use cases of quantum computing is solving optimisation problems. Optimisation problems are difficult to solve due to incomplete, uncertain data, huge computational space. To address these problems the variational quantum eigensolver (VQE) [PMS+14] and the quantum approximate optimisation algorithm [FGG14b] have been used in different variations. These algorithms are used for finances [HGL+23], even trying to solve NP-hard problems [ACH23]. To run these algorithms on a quantum computer they have to be optimised according to the quantum computer topology. There is a need to find suitable algorithms that would prepare optimisation algorithms to run on the desired quantum computer in such way that it would require the least amount of modifications and has a low error probability.

Building on this context, the research objects are quantum optimisation algorithms and quantum computing topologies.

The aim of the work is to develop and validate a method for optimisation and construction of the quantum algorithms on different quantum computers topologies by focusing on improving requirements on number of qubits and computational scheme complexity.

Objectives:

1. Make a scientific literature review and identify most common quantum computer topologies, typical algorithm construction methods and scheme optimisation methods.
2. Run the empirical experiments and theoretical investigation on applying scheme optimisation methods to identified quantum computer topologies.
3. Formulate the optimisation success criteria (CNOT gate count, circuit depth) and propose generalisation of scheme optimisation on different topologies.
4. Investigate proposed approach properties under additional error correction complexity.
5. Validate experiments on real quantum computers.

Hypothesis - generalisation of scheme optimisation on different topologies is feasible under the success criteria (number of CNOT gates, scheme depth).

1. Theoretical foundations

1.1. Quantum circuit

1.1.1. Qubit

Classic computing bit can be represented as a binary digit of 0 or 1. Unlike a classical bit, a qubit (quantum bit) has two basic states $|0\rangle$ and $|1\rangle$. This means that qubit can be in a superposition state of both 0 and 1 simultaneously. This superposition state is shown as a $a|0\rangle + b|1\rangle$, where $|a|^2 + |b|^2 = 1, a,b \in \mathbb{C}$. The values $|a|^2$ and $|b|^2$ represents a probability of the qubit state collapsing to $|0\rangle$ or $|1\rangle$ upon measurement. A qubit is often initialised to the $|0\rangle$ state and then transformed by a quantum circuit.

1.1.2. Quantum gates

Quantum gates are used to change the state of qubits. They are fundamental operational units in quantum computing. Quantum gates can consist of a single qubit or multiple qubits. Gates like Hadamard, Pauli-X are single qubits gates, whereas CNOT or SWAP are two-qubits gates.

Each quantum computer supports only a subset of quantum gates. The unsupported quantum gates have to be transpiled to the known ones. For example, IBM Heron processor supports CZ, ID, RZ, SX, X gates[1]. Clifford + T is one of the universal gate sets that has recently gained its popularity [FMM+12; Got98]. A universal gate set is a set of gates from which any unitary operation can be constructed or approximated to arbitrary accuracy [NC10].

1.1.3. Quantum circuit

To represent a quantum algorithm, the quantum circuit concept is used. In the quantum circuit, each line represents a qubit, boxes on the qubit lines represent operators. The execution order of a quantum circuit is from left to right. A layer in a quantum circuit is defined as a set of non-intersecting and parallel executable quantum gates. The number of layers represents quantum circuit depth. In the end of the circuit, in most cases the measurement is used. It collapses qubit's superposition into a classical state.

1.2. Quantum computers topology

Quantum computers can be built in several ways. The most popular and known ways:

- Superconducting quantum computers [WBC+21]
- Trapped-ion computers [MK13]
- Photonic quantum devices [PMS+14]
- Neutral atoms [WDE+23]

[1] https://docs.quantum.ibm.com/guides/processor-types

Each of them have their own advantages and disadvantages. Due to their differences, they also differ in the quantum computer topology. Quantum computer topology is often represented as a graph. Currently, the most investigated quantum computer topology is superconducting quantum computers. They are selected for their good scalability. Also, IBM and Google provide quantum computers, which are superconducting ones, that can be used by any ordinary person who is interested in quantum computing. Example of the IBM Heron quantum computer topology is in Figure 1. Major drawback of superconducting quantum computers is that each qubit is only connected to a subset of qubits in the quantum computer. Due to this constrain, not every multiple qubit quantum gate that is placed on a quantum circuit can be executed on this type of quantum computers. It cannot be done because the qubits that are needed for the multi-gate are not connected in the quantum computer. This problem is called qubit mapping problem.

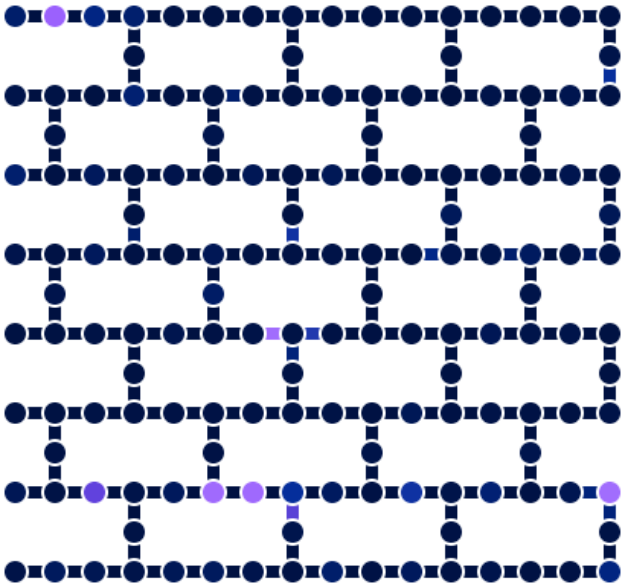

Figure 1. IBM Heron quantum computer topology. The colour in this graph shows for the nodes the readout assignment error gate error rate and for the edges the gate time (ns). The darker it is, the better. Source: IBM

1.3. Qubit mapping problem

Quantum circuit mapping is considered as a fundamental mechanism in the current NISQ era [LDX19a]. The goal of solving this problem is to transform the quantum scheme that it would be runnable on the selected quantum computer.

1.3.1. Definition

- **Input:** quantum algorithm circuit (can be also called a logical circuit), quantum computer topology (can also by called physical circuit).
- **Output:** transformed quantum algorithm circuit that can be executed on the provided quantum computer.

Qubit mapping problem is split into two parts:

1. Initial qubit mapping.
2. Routing qubits by using two-qubit gates.

Example of qubit mapping problem is presented in Figure 2.

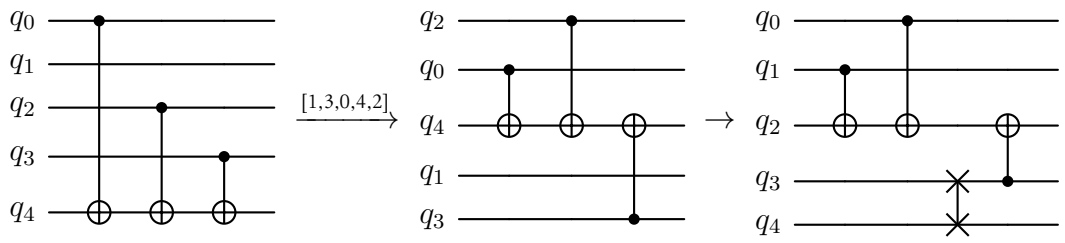

Figure 2. Qubit mapping problem example. The quantum scheme on the left is the initial quantum scheme, the middle one represents the initial qubit mapping found and the right represents the routing qubits by adding the SWAP operators. On the first arrow the mapping of the qubits is represented.

1.3.2. Initial qubit mapping

Initial qubit mapping is a process when positions are selected to map virtual (sometimes called logical) qubits of the circuit to physical qubits [TFX+23]. While the routing method is often responsible for introducing a number of gates, the effectiveness of routing is fundamentally determined by the initial qubit allocation. Qubit allocation methods are chosen either to minimise the depth of the circuit or to maximise the success rate of the circuit. The success rate of the circuit - minimising the number of the SWAP operators used in the routing phase. Not only the number of SWAP operators are taken into consideration but also the selection of qubits that have the highest fidelity - lowest error rates and longest coherence times. This is crucial for ensuring that the desired quantum circuit will return predictable results.

1.3.3. Routing qubits

In the perfect scenario, after initial qubit mapping the circuit would be executable on a selected quantum computer with no additional changes to the quantum circuit. However, due to the nature of the initial qubit mapping algorithms and the topology constrains, in most cases additional operators insertion will be needed. SWAP operators are the most commonly used quantum gates to solve this problem. SWAP operator swaps two-qubit states. By inserting SWAP operators this allows the needed two-qubits to communicate with each other. Each SWAP operator consists of three CNOT gates (see Figure 3).

The number of SWAP operators used has to be minimised. Each quantum gate added to the circuit adds additional noise to the circuit and each qubit has its own coherence time (the time for how long it can maintain its state). This can result into incorrect quantum algorithm results after measuring.

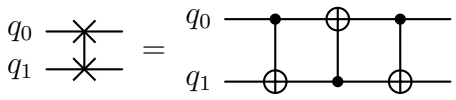

Figure 3. SWAP operator implementation.

1.3.4. Major types of qubit mapping problem solutions

Solutions to the qubit mapping problem can be split into two parts [LD24]:

- Approaches that return optimal solutions but they can only be used for a small scale quantum computers and algorithms.
- Heuristic solutions that cannot ensure that provided solution is optimal. Search space is reduced in these type of algorithms.

1.3.5. NP-hard problem

Qubit mapping problem is a NP-hard problem [IKK+25]. In this article it was proven that the qubit mapping problem is at least as difficult as another NP-hard problem called Optimal Linear Arrangement (OLA) [GJS76]. The challenge of finding the minimum number of SWAP gates for qubits on a line is fundamentally equivalent to solving OLA which involves arranging items in a line to minimise the total distance between item pairs. No efficient algorithm is known for OLA, so the qubit mapping problem is also a NP-hard problem.

1.3.6. State-of-the-art algorithm (SABRE)

SABRE (SWAP-based BidiRectional heuristic search algorithm) [LDX19b] is considered as a state-of-the-art algorithm for solving qubit mapping problem. Algorithm steps:

1. From the quantum circuit the Directed Acyclic Graph (DAG) is created that represents the dependencies between the two-qubit gates. The ”front layer” set is used which is a set that stores all two-qubit gates that have no pending dependencies and can be executed.
2. Algorithm loops until all gates are executed. For each iteration:
   (a) Checks if any two-qubit gates can be executed with the current qubit mapping. If there is some, the ”front layer” is updated with these gates.
   (b) If there are no gates in the gates set, then the list of possible SWAP operators that interact with at least one qubit in the set is generated.
   (c) Each SWAP operator is evaluated by using a heuristic cost function. The look-ahead approach is used where not only the current step is taken into the consideration but how the SWAP gate will impact future steps.
   (d) SWAP with the lowest score is selected and added to the circuit.

   This algorithm is valued for its scalability, good initial mapping results. It is also implemented in the Qiskit library.

1.4. Noise

In current quantum computer era it is difficult to obtain consistent and correct results of running the algorithm. This is the reason why this era is called Noisy Intermediate-Scale Quantum era [Pre18]. Error types that have impact on qubit mapping problem results:

1. Crosstalk errors [NT21]. This noise source can corrupt qubit state.
2. Single qubit error rate [Ale19]. This shows the probability that a single-qubit gate will introduce an error.
3. T1 (Relaxation) error [NC10]. Qubit is losing its energy and decaying from the state $|1\rangle$ to $|0\rangle$.
4. T2 (Dephasing) error [NC10]. Qubit is losing its quantum phase information.

1.5. Quantum optimization problems

Quantum optimisation problems represents a new way for solving complex computational problems where finding the best solution is difficult. Many different fields including logistics, chemistry can make use of finding the minimum (or maximal) value of a complex cost function over a vast search space [BBC+24]. The main advantage of quantum optimisation algorithms is that superposition and entanglement can be used to explore search space faster. Two of the leading algorithms in this class are Variational Quantum Eigensolver (VQE) [PMS+14] and Quantum Approximate Optimization algorithm (QAOA) [FGG14a].

1.5.1. Variational Quantum Eigensolver (VQE)

The Variational Quantum Eigensolver (VQE) is a hybrid quantum-classical algorithm that is designed to find the lowest energy state of a physical system [PMS+14]. Finding the ground state energy of a molecule or material is an important problem in quantum chemistry. Algorithm steps:

1. A Hamiltonian is created for the physical system that is going to be studies. Hamiltonian is a quantum operator that describes the total energy of a system. The initial parameters θ are selected for the ansatz. The ansatz example for two-qubits can be seen in Figure 4
2. The quantum processing unit (QPU) receives the θ parameters. The quantum gates are configured according to the provided parameters and the trial quantum state $|\psi(\theta)\rangle$ is set up.
3. Then the expected energy $\langle\psi(\theta)| H |\psi(\theta)\rangle$ is measured of the prepared state by running the ansatz. This is done by breaking the Hamiltonian into a Pauli operators. The state is measured many times to get sufficient statistics.
4. Classical optimisation algorithm takes the measured energy value from the QPU and calculates new set of parameters θ.
5. The steps 2-4 are repeated until the measured energy value has reached minimum value. The lowest energy from VQE is the approximation of the ground state energy.

Possible use cases of VQE algorithm:

- Quantum chemistry [PMS+14]. Calculation of molecular energies that can be used for drug discovery.
- Quantum Machine Learning [UKB20]. VQE can be used to classify phases of matter.

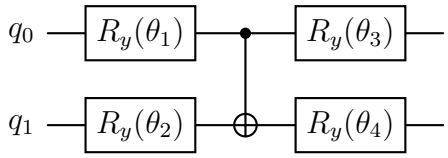

Figure 4. A simple two-qubit VQE ansatz circuit.

1.5.2. Quantum Approximate Optimization Algorithm (QAOA)

The Quantum Approximate Optimization algorithm (QAOA) [FGG14a] is a hybrid quantum-classical algorithm that is designed to find approximate solutions to combinatorial optimisation problems. Algorithm steps [BBC+24]:

- Encoding a problem into a cost Hamiltonian. A Hamiltonian is a quantum operator that describes the total energy of a system. It is constructed in a way that its lowest energy state corresponds to the optimal solution of the problem,
- Designing the quantum circuit that is called the ansatz. The algorithm starts with the qubits in the superposition by applying a Hadamard gate. Then the ansatz consists of p repeated layers:
  - The cost operator.

$$U_C(\gamma) = e^{-i\gamma H_C} = \prod_{i=1,j<i}^{n} R_{Z_i Z_j}(-2w_{ij}\gamma)$$

    This operator is controlled by a trainable parameter γ. It applies a phase to each possible solution based on its cost. It marks good solutions with a different phase than bad ones.
  - The mixer operator

$$U_M(\beta) = e^{-i\beta H_M} = \prod_{i=1}^{n} R_X(-2\beta)$$

    This operator is controlled by a trainable parameter β and it is crucial for exploration. It is built from a Mixer Hamiltonian (H_M. In standard QAOA is $H_M = \Sigma X_i$. The main property is that H_C and H_M do not communicate. This allows this mixer to transition the quantum state between different possible solution. This helps to explore the search space.
- The hybrid algorithm part.
  1. Parameters initialisation. Initial set of $2p$ parameters is set (the set of γ and β parameters for each layer.
  2. Parameters are sent to the quantum processing unit where it prepares the initial state and runs p-layer ansatz circuit by using these generated parameters.

3. The final quantum state is measure repeatedly due to the noise. This gives a probability distribution over all possible solutions.
4. Cost is calculated from the measurement results.
5. Classical computer is used with optimisation algorithm like COBYLA [Pow94] to suggest a new set of parameters based on the calculated cost to produce better cost value.
6. Steps 2-5 are repeated until the cost value converges or a maximum number of iterations is reached.

- After optimising the parameters, the circuit is run for the last time with these parameters and the most frequently received bit string is a approximate solution to the problem.

Example of the circuit can be found in Figure 5.

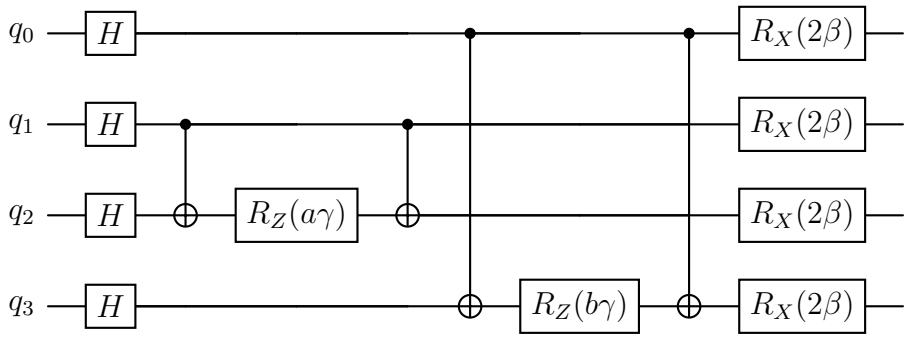

Figure 5. A four-qubit QAOA circuit example.

Possible use cases of the QAOA algorithm [BBC+24]:

1. Graph problems like MaxCut [FGG14b].
2. Maximum Independent Set (MIS) [CK19]: Find the largest set of non-adjacent vertices that no two vertices share an edge.
3. Travelling Salesperson Problem [RMX+20]: Find the shortest path to visit all the nodes.
4. Object Detection in AI [LG20]. By using QAOA it can improve the speed of identifying objects in images.
5. K-means clustering problem as a QUBO problem [MWB+22].
6. Protein Folding [MMJ+22]. Allows to predict the three-dimensional structure of a protein from its amino acid sequence

2. Systematic literature analysis

Systematic literature review is a transparent and systematic process to define a research questions, to access the quality of the sources and to analyse the findings qualitatively or quantitatively [AHD+11]. In this work systematic literature review is used to get knowledge about how the qubit mapping problem and its appliances on the quantum optimisation algorithms.

2.1. Selected method

In this work the systematic literature review that was proposed by Kitchengam and Charters is used [KC07]. This is the method that is proposed for software engineering literature review. It was chosen as it is a popular option for conducting literature review. Literature review steps [KC07]:

1. Planning the review:
   - Identifying a need for a review
   - Commissioning a review
   - Specifying the research questions
   - Developing a review protocol
   - Evaluating a review protocol
2. Conducting the review
   - Identification of research
   - Selection of primary studies
   - Study quality assessment
   - Data extraction and monitoring
   - Data synthesis
3. Reporting the review
   - Specifying dissemination mechanisms
   - Evaluating the report

2.2. Literature analysis planning

2.2.1. Identifying a need for a review

According to the provided literature review methodology it is needed to check if a review is necessary and review any existing systematic reviews of the same domain [KC07]. At the time of the writing no systematic literature review was done on taking a look on qubit mapping problem by taking a primary focus on quantum optimisation algorithms. The closest review was done by Chinese scientists who compared possible algorithms for quantum compilers [ZWY+25]. But in their work they are analysing concrete solutions for specific quantum computer types. So there is a need to see what are the newest ways of solving qubit mapping problem, how it can be applied for quantum optimisation algorithms, how the performance and how the current approaches can be improved.

2.2.2. Selected analysis questions

Choosing the research question in the most important part of systematic literature review [KC07]. These research questions model how the review will be conducted. According to the topic of the research, the following questions were selected:

Q1. What are the most recent developments and trends in quantum circuit optimisation specifically related to qubit mapping?

Q2. How to measure performance of algorithms that solve qubit mapping problem?

Q3. How can quantum optimisation algorithms be further optimised compared to ordinary quantum algorithms?

Q4. How does the topology of a quantum computer influence the performance of quantum circuit optimisation algorithms?

Q5. What are the identified research gaps and unresolved challenges in current literature regarding qubit mapping and circuit optimisation?

2.2.3. Selected databases

To perform a literature analysis, the sources of the research articles have to be selected. Criteria for the database:

- It has to be reputable.
- Contain research articles about qubit mapping problem.
- Have a proper search engine.

Selected databases that have met the criteria:

1. IEEE Xplore
2. Springer
3. ACM Digital Library
4. Web Of Science

2.2.4. Study selection criteria

The systematic literature review uses study selection criteria that defines what articles will be included and excluded.
Inclusion criteria:

1. Date of publication - from 2023. This was selected to see the newest trends in solving qubit mapping problem.
2. Language of the work - English.
3. The full text of the study is accessible.

Exclusion criteria:

1. Date of publication - before January 1, 2023.
2. Language of the work - other than English.
3. The full text of work cannot be obtained.

2.2.5. Study quality characteristics

According to the chosen systematic literature review method, the study quality characteristics have to be defined. These characteristics allows to filter out unreliable, unrelated studies that can have impact on the research quality. The selected study quality characteristics:

QC1. The central theme of the study is about quantum circuit optimisation and qubit mapping problem.

QC2. The study compares, proposes or analyses algorithms that are used for solving the qubit mapping problem (it can be part of the work but it has to have proper analysis).

QC3. The study explicitly describes, proposes or analyses metrics, benchmarks that were used for evaluating the performance of qubit mapping problems.

QC4. The study discusses identified research gaps or unresolved challenges related to qubit mapping problem.

By using these quality characteristics studies that provided deep contribution to the field will be selected. Also, the studies that detail the metrics and benchmarks used can be more reliable and they can prove that they proposed method is better in some regard than other approaches. Studies that provide research gaps or challenges can help to generate the ideas for the future work, how the current approaches can be improved and, moreover, it shows that researchers are aware of their method limitations.

2.2.6. Screening the studies

Following the systematic review methodology, every study will be assessed to the provided quality characteristics. The score will be given corresponding to the number of quality criteria it will satisfy. The assessment is binary - it either fully satisfies a criterion (1 will be given) or it does not (0 will be given). This means that each paper can receive a total score up to 4 points. By following this grading methodology, it was chosen that the studies that have scored 3 or more points will be marked as an included in the final data synthesis.

2.2.7. Defined search string

Search phrase that will be used to find the studies:

("quantum circuit optimization" **OR** "quantum circuit optimisation" **OR** "qubit mapping problem" **OR** "qubit allocation")

2.3. Conducting a review

2.3.1. Gathering and filtering results

The search phrase has been used on the selected databases. At first stage the study selection criteria has been applied. Then at the second stage the studies abstracts and their titles were scanned to verify that the selection criteria is passed. At the end the duplicates were removed. The flow results are represented in Figure 6.

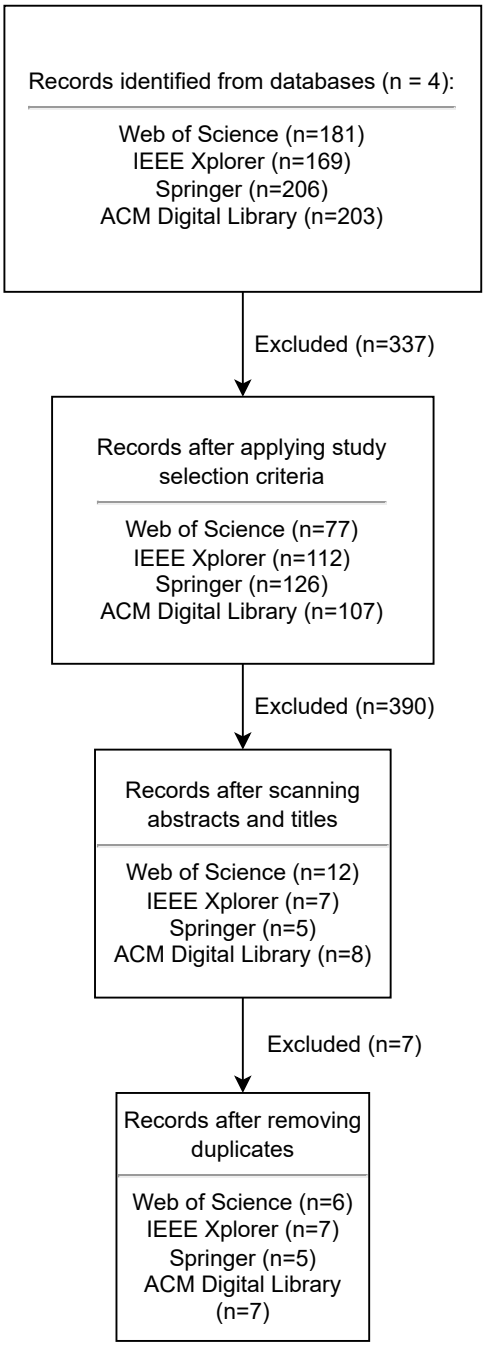

Figure 6. The flowchart illustrating the systematic review process for selecting studies. It shows the initial number of records and the subsequent filtering at each stage.

The sources that have been added to the quality characteristic evaluation are provided in Table 1.

Table 1. Sources that have been added to the quality characteristic evaluation

Database	Sources
Web of Science	[FZW+23; JKP24a; LLW+24; LNC+23; MZ23; NT23]
IEEE Xplore	[AV24; CSS+25; KSZ+23; SIM+23; TFX+23; YLJ24; YYH+24]
Springer	[BDS25; JDX24; JFD+25; LZZ+24; TSL23]
ACM Digital Library	[CYK+24; KDS+24; LD24; PBW23; TCJ24; UL23; WB23]

2.4. Studies analysis

2.4.1. Quality characteristic evaluation

Quality characteristic evaluation for selected research articles is presented in Table 2. The score of the quality characteristic satisfaction is binary: 1 - if criteria is fully satisfied, 0 if not. Sources that have 3 or more are included into studies analysis.

Table 2. Quality characteristic evaluation for selected research articles

Database	Paper title	QC1	QC2	QC3	QC4	Sum
Web of Science	A doubly stochastic matrices-based approach to optimal qubit routing	1	1	1	0	3
Web of Science	Improving the performance of digitized counterdiabatic quantum optimization via algorithm-oriented qubit mapping	1	1	1	0	3
Web of Science	Optimization of Qubit Mapping Model Based on Deep Q Network	1	1	1	0	3
Web of Science	Single-Qubit Gates Matter for Optimising Quantum Circuit Depth in Qubit Mapping	1	1	1	1	4
Web of Science	Suppressing Quantum Circuit Errors Due to System Variability	1	1	1	1	4

Springer	PSO inspired global neighbourhood based Qubit mapping: a new approach	1	1	1	0	3
Springer	A binary integer programming-based method for qubit mapping in sparse architectures	1	1	1	0	3
Springer	QM-DLA: an efficient qubit mapping method based on dynamic look-ahead strategy	1	1	1	1	3
Springer	Qubit Mapping Based on Tabu Search	1	1	1	1	4
IEEE	Effective and Efficient Qubit Mapper	1	1	1	1	4
IEEE	TRIM: crossTalk-awaRe qubIt Mapping for multiprogrammed quantum systems	1	1	1	0	3
IEEE	Machine-Learning-Based Qubit Allocation for Error Reduction in Quantum Circuits	1	1	1	1	4
IEEE	PauliForest: Connectivity-Aware Synthesis and Pauli-Oriented Qubit Mapping for Near-Term Quantum Simulation	1	1	1	1	4
IEEE	Quantum Circuit Mapping Using Binary Integer Nonlinear Programming	1	1	1	0	3
IEEE	Towards Effective Local Search for Qubit Mapping	1	1	1	1	4

IEEE	Towards Fidelity-Optimal Qubit Mapping on NISQ Computers	1	1	1	1	4
ACM	An Ising-based Model for Qubit Mapping	1	1	1	1	4
ACM	Exploiting the Extended Neighborhood of Hexagonal Qubit Architecture for Mapping Quantum Circuits	1	1	1	1	4
ACM	MQT QMAP: Efficient Quantum Circuit Mapping	1	1	1	1	4
ACM	On Optimal Subarchitectures for Quantum Circuit Mapping	1	1	1	1	4
ACM	Satisfiability Modulo Theories-Based Qubit Mapping for Trapped-Ion Quantum Computing Systems	1	1	1	0	3
ACM	Robust Qubit Mapping Algorithm via Double-Source Optimal Routing on Large Quantum Circuits	1	1	1	1	4

2.5. Literature analysis results according to the selected analysis questions

2.5.1. Response to question 1

The question 1 was: *What are the most recent developments and trends in quantum circuit optimisation specifically related to qubit mapping?*

Various techniques are used to solve the qubit mapping problem. Proposed techniques:

Mathematical optimisation. Algorithms that formulate qubit mapping problem as a formal mathematical problem and solves them by utilising specialised solvers. They often pro-

vide optimal solutions but are limited to small quantum computers and circuits due to their computational complexity.

- Encoding the problem to binary integer programming problem (BIP) [JFD+25]. It is proposed for sparse architectures. Algorithm steps are as follows:
  1. Circuit is sliced into smaller sub-circuits. Slicing is done in triangle patterns.
  2. Initial qubit mapping is done by focusing only on first interactions between any two-qubits. It finds the best match between initial interaction graph and quantum computer topology while prioritising mappings that place qubits on physical locations with high-fidelity connections.
  3. Then the each slice is converted into BIP model. A binary matrix M is created that represents assignment of logical to physical qubits. SWAP gates are modelled as a row transformation.
  4. Shortest path is used to find sequences of SWAP operators to connect two non-adjacent qubits.
  5. The goal is to maximise the fidelity of the compiled circuit. The function is a weighted sum where each term is the fidelity of a chosen path.
- Solving qubit mapping problem as a MaxSAT problem [WB23]. The scientists have proposed QMAP tool which can be used to solve the qubit mapping problem. Solving it as a MaxSAT problem with SMT (Satisfiability Modulo Theories) solvers returns optimal results. However it can only be used until 8 qubits because of the NP-hardness.
  Observations: While the exact solution algorithm will find the best result, it is too slow to use for the current and upcoming quantum computers.

- Another research that is taking a look into a subcircuits is a SMT-base qubit mapping algorithm [TCJ24]. This approach was selected for fully connected graphs that are in trapped-ion quantum computers. The problem solved is not by adding a SWAP operators but reducing overlapping of using the qubits at the same time. SMT solves a formal logic problem where:
  – Each logical qubit of the circuit must be mapped to a unique physical qubit
  – If two gates operate on the same qubit, they must be executed in the correct logical order.
  – If the physical gates position overlap, they must be assigned to different time steps.

  Valid mapping is searched at the each time step. Divide and conquer approach was used from splitting large circuits into smaller ones, finding an optimal mapping for the subcircuit and merging the solution into one. This allows to a speedup in the computation time for larger circuits.
  Observations: SMT-based method finds good quality solutions, but this solution is only applicable to trapped-ion or the quantum computers that have qubits fully connected.

- Quadratic Unconstrained Binary Optimisation problem is solved [UL23]. Scientists have

used The Fujitsu Digital Annealer [Fuj25]. This parallel hardware accelerator is used for solving this type of problems. It is based on simulated annealing but it has been optimised to used CMOS hardware for it.

Observations: In reason that this research used specific hardware that is not that accessible, we will ignore it for the candidate of the solution.

Local search algorithms. Algorithms that start with potential or random generated solution and tries iteratively improve the solution by making local changes.

- Isomorphic Graphs [JDX24]. In-Memory Subgraph Matching (IMSM) algorithm can be used to find an initial qubit mapping and Tabu search for qubit routing [JDX24]. Algorithm steps:
  1. IMSM finds the logical interaction map that is isomorphic to a subgraph of the quantum computer topology.
  2. Then the proposed Tabu search is used. Tabu search helps to jump out of the local optimum. For each non-executable gate the algorithm identifies what SWAP operators can be used. To connected qubits SWAP operator is placed on the shortest path between them. To choose the best SWAP from the candidates, this algorithm uses a look-ahead evaluation function. The impact of next few layers is taken into consideration.
  3. Once the best SWAP is selected and applied, its reverse operation is added to the Tabu list. This prevents algorithm of going back and it helps to avoid getting stuck in a loop.

  . *Algorithm evaluation*: While this algorithm performs quite fast, it is a heuristic method and it might not return the most optimal solution. Also this algorithm is sensitive to the physical qubits graph density, the denser the topology, the less effective the initial qubit mapping is.

- Using a local search algorithm which is dedicated to solve qubit mapping problem [CSS+25]. This algorithm is called EffectiveQM. This EffectiveQM has two modes:
  - Exploitation Mode - continuously tries to improve the current best solution by applying the best possible local changes. To decide which operation is the best, the Potential-Guided Scoring Function is used. It takes into consideration two factors: 1) immediate benefit - how it will improve circuit now (by calculating how the distance is reduced); 2) potential benefit - what is the best possible immediate benefit for the next move. If the algorithm does not find better solution for δ times, it concludes that it is stuck in the local optimum. Then algorithm switches to Exploration mode.
  - Exploration Mode - it abandons the current path and generates a new random starting mapping and the begins process from a completely different part of the solution space.

  This two-mode local search design is recognised as having a good success for solving variety of NP-hard optimisation problems.

  Observations: Fewer additional two-qubit gates are added with this approach, algorithm has a quite clear structure, local optimal solution is escaped with this approach but it

has slower runtime and it is difficult to choose the appropriate hyperparameters for the specific quantum circuit and quantum computer topology.

- Particle Swarm Optimization (PSO) approach combined with a global neighbourhood-based heuristic [BDS25]. Each particle represents a possible qubit mapping. By updating velocity fitness is calculated and the solution is the one with the lowest fitness value. Algorithm steps:
  1. Quantum circuit is represented as a Control Flow Graph where nodes are qubits and edges are CNOT operators.
  2. The logical qubit with the highest degree is selected. Degree is to how many qubits it is connected to.
  3. This qubit is mapped to the physical qubit that also has the highest degree.
  4. Then the neighbours of this logical qubit are also mapped to the neighbours of the physical qubit.
  5. Then the search for the best solution starts. The initial swarm is filled with the solutions from the set that has been used before. To evaluate how good the mapping is (a particle) the fitness function is used: Total SWAP count is summed with sum of degree differences.
  6. This mapping process continues until the number of iterations is reached.

  Observations: Algorithm is fast, but it was not tested on the bigger quantum circuits, performance depends on the algorithm parameters and it is a high probability that it will not return the best result due to the algorithms nature.

Using look-ahead strategies. Algorithms that focuses on using greedy algorithms to modify a circuit by looking ahead at upcoming gates to make decisions about SWAP insertions.

- A* using QMAP tool which can be used to solve the qubit mapping problem [WB23]. To find a solution for bigger topologies, A* algorithm is used as a heuristic mapper. It traverses the search space of the mapping problem.
  Observations: Heuristic approach is more promising but isomorphism is not utilised in this work and this can lead to exploring redundant possibilities.

- Circuit is converted into Directed Acyclic Graph and sliding window search algorithm is used [FZW+23]. The authors have proposed SWin algorithm that works in this manner:
  1. Only a small, fixed-depth section of the upcoming gates in the circuit is considered.
  2. A* algorithm is used to find the best sequences of gates and SWAPs to minimise depth for the specific window.
  3. Window slides forward and the calculation process is repeated on the remaining of the circuit.

  In the paper VQE algorithm was used for results comparison which is related to this work.
  Observations: This approach produces shallower circuits, is quite efficient, but it will not

produce the most optimal solution, it is sensitive to a quantum computer topologies that have a very high connectivity because the number of possible SWAP operations is increasing. Also, the parameters have to be adapted according to the quantum computer topology.

- Dynamic multi-window look-ahead strategy with a heuristic cost function (MSPE) to intelligently insert the minimum necessary SWAP gates [LZZ+24]. Algorithm steps:
  1. Initial qubit mapping. Qubit Pair Interaction metric is calculated for each qubit pairs that gives more weight to the qubit pairs that interact more frequently and earlier in the circuit. Another metric is calculated - Physical Connectivity Strength (PCS). PCS is the sum of the number of its first-neighbouring qubits and the sum of its second-neighbouring qubits. Algorithm then maps the highest-priority logical qubit to the physical qubit with the highest PCS.
  2. Routing with SWAP operators. The algorithm traverses the DAG (Directed Acyclic Graph). To choose the best SWAP it uses a cost function that evaluates the immediate increase in circuit depth and the sum of distances for future gates with a look-ahead window.

  Observations: This algorithm reduces SWAP gate count, provides results that compares with SABRE. The calculations are quite slow and it might be a problem with bigger quantum computer topology and the implementation of the algorithm is quite complex.

- Depth-First Search can be used for initial qubit mapping [CYK+24]. Algorithm steps:
  1. Sorting qubits in DFS order and mapping them on quantum computer topology. According to this work initial mapping is more important for shallow circuits. As for growing circuit complexity it does not have big importance as the routing and scheduling takes a more significant part, according to them.
  2. For routing Dual-source Dijkstra (they call it Doustra) is used. Path from both connected qubits is calculated and the goal is to find a target pair where the logical qubits can meet. The cost of path is not also a number of SWAPs but also the occupied time on the qubit is taken into consideration.
  3. The scheduler is used to decide the order in which to execute the prepared gates from the circuits dependency graph. Two strategies are proposed:
     - Limitedly Exhaustive - for smaller circuits all possible gate sequences are tested up to a certain depth.
     - Shortest Path Estimation - For large circuits the cost of executing a gate is calculated by combining shortest path and the current occupied time.

  Observations: This is probably the best scaling algorithm of all presented here and it makes this algorithm quite future proof. But this algorithm is sensitive to the circuit structure. For highly repetitive and structured circuits this greedy approach might get stuck on the local optimal point.

Decomposition strategies. Algorithms that break the problem into smaller ones by partitioning the quantum circuit or the quantum computer topology graph.

- By using subcircuits [PBW23] QMAP tool is generating a subarchitectures for a specific circuit [PBW23]. According to the researchers n-qubit circuit should only consider subarchitectures that have similar amount of qubits. In this paper it is suggested to have more than n qubits because it can allow to have shorter paths between qubits. Also multiple subarchitectures are generated for a circuit and by using a routing algorithm the best one is selected.

  Observations: While it returns more optimal solution, this approach only has acceptable optimisation time for devices smaller than 30 qubits which is too small for this research.

- Subgraphs are searched and then initial qubit mapping algorithm is used [SIM+23]. Algorithm works as follows:
  1. Search for the best physical qubit subgraph on the quantum circuit to execute the circuit. By using hardware's calibration data it calculates the total error score for the subgraph and selects one with the lowest error rate.
  2. Mapping the busiest logical (from the quantum circuit) qubits to the most central physical qubits. In the chosen subgraph it is calculated which physical qubits have the shortest average distance to all other qubits. Then it maps the logical qubits with the highest interaction frequency to the physical ones with the highest centrality.
  3. Heuristic cost function is used to calculate the best SWAP candidates.

  This paper uses brute-forcing all sub-topologies.

  Observations: Even though this will provide reliable subgraph (or initial mapping), this approach is inefficient and will be a bottleneck for a bigger quantum computer topologies because it checks for the all sub-graphs.

Specialised for quantum optimisation algorithms.

- Suzuki-Trotter decomposition is used for approximating unitary time evolution operators [JKP24a]. Algorithm is called AOQMAP. Four variations of DC-QAOA algorithm are explored in this paper. Algorithm steps:
  1. AOQMAP begins with two-qubit terms of DC-QAOA Hamiltonian. For example, ZZ, YZ.
  2. Then executable circuit fragments are generated for common hardware topologies (linear, T-shaped). This is done by inserting SWAP operators. Swap layers and their mirror symmetry is used to construct gate sequences that are efficient for DC-QAOA algorithm.
  3. Then the circuit is optimised according to the problem:
     - For problems with sparse connectivity (like MaxCut on non-complete graphs), the initial qubit mapping is optimised to minimise the SWAP operators.

– For fully-connected problems, initial qubit mapping is selected that has the best performance after classical optimisation loop.

4. Before execution a cost function is used to select best physical qubits on the target device.
5. By using classical optimisation routine (for example, COBYLA [Pow94]) the best variational parameters are selected for the generated and mapped circuit.

Observations: This approach greatly reduces amount of added two gate operators. More, this approach is created for optimisation algorithms like QAOA which is the part of this research. The negatives about this approach that on sparse graphs the better performing initial qubit mapping is needed. Additionally, it depends on the set of subtopologies and there is a need to use a newer ones for our current most popular quantum computers.

- PauliForest [YYH+24]. Pauli-orientated, static approach that is designed for quantum simulation kernels. Algorithm steps:
  1. From all Pauli strings in the quantum circuit the feature matrix is built. In the matrix row represents a Pauli string and a column represents a logical qubit. Value 1 is set if it has a non-identity operator in the string, else 0.
  2. Correlation score between pair of qubits is calculated. The dot product of their respective columns is taken in the feature matrix.
  3. The most used logical qubit (one that has most 1 in a column) is selected and it is mapped as a central node.
  4. The remaining qubits are mapped one by one. In each step the qubit with the highest correlation with all the qubits that are already connected is selected. Once this qubit is chosen, the algorithm searches for the best possible physical qubit location. The cost is calculated by summing the shortest distances in quantum computer topology between the candidate node and all other qubits that it interacts.

  Observations: This algorithm is suitable for quantum optimisation algorithms that use Pauli Z gates. This allows to reduce SWAP count for these algorithms. But this is algorithm is quite sensitive to quantum computer topology because the sparser the graph, the less beneficial this approach is.

Machine learning. Algorithms that use machine learning models to learn a policy for making mapping and routing decisions.

- Deep Q-Network that uses a deep neural network is used to implement Q-Learning algorithm [LLW+24]. Deep Q-Network is a type of reinforcement learning that is used in this work to make sequential decisions. Algorithm steps:
  1. Simulated annealing is used to find an initial qubit mapping.
  2. DQN model with optimised SWAP insertion strategy is used. They call this optimised SWAP insertion SA-DQN. Heuristic cost function is used for selecting where to use

SWAP operators. During this process, not only current step is evaluated but also how it will effect future ones.

Observations: It combines two optimisation techniques but it is unclear how it compares with SABRE, also, it has complex hyperparameter tuning, model will have to be trained.

- Another user neural network - GNN (graph neural network) based Q-network [TFX+23]. This strategy was called GNAQC. The quantum computer topology is fed to GNNs. GNNs are used to analyse the error rates and connectivity of the physical qubits. The matrix representation of quantum circuit is processed by a standard feed-forward neural network. The outputs from both branches are combined, along with the current state of the mapping. This information is fed into a Q-Network. Q-Network predicts the optimal initial mapping by outputting a score for every possible placement action. This model takes decisions by the reinforcement learning. Reward calculation:
  1. Chosen layout is selected.
  2. Qubit routing is done by using SABRE algorithm.
  3. Final circuit is simulated by using quantum computer calibration data.
  4. The output distribution is compared to error-free output distribution.
  5. The fidelity between these two distributions is a reward.

  Observations: This work shows that initial qubit mapping improves the overall performance, because the fidelity has increased by 12,7 % compared to regular SABRE algorithm. As for the research paper before, this approach also requires training, also it relies on an accurate simulator. Moreover, this algorithm proposes only the new approach for initial qubit mapping, routing part is not updated.

Novel continuous optimisation approaches. Algorithms that changes the routing problem by using new approaches or ideas.

- Doubly stochastic matrices are used to transform SWAP insertion task into a continuos optimisation problem [MZ23]. Doubly stochastic matrix is a square matrix with non-negative entries where every row and every column sums to 1. According to Birkhoff-von Neumann theorem and doubly stochastic matrix can be written as a weighted average. Smooth Swaps (SSWAP) concept was introduced in this work where if the $\theta = 0$ then no SWAP is used, if $\theta = \frac{\pi}{2}$ then operator is SWAP. Value in between means partial swap in the mathematical space. To solve this optimisation problem a rolling horizon approach is used by considering a small window of upcoming circuit layers.

  Observations: Circuit depth is significantly reduced but this algorithm is quite difficult to comprehend and it has computationally intensive calculations.

Problem extensions. Algorithms that solve a part of qubit mapping problem or they are adapted for a new context (for example parallel quantum circuits execution) or act as a additional step to an existing qubit mapping solution.

- An interesting approach was taken to take into consideration single-qubit gates [LNC+23]. According to this research adding SWAP operator after a long sequence of single-qubit gates results in doubling the circuit depth. Algorithm steps:
  1. Storing single-qubit into the buffer and is only executed when a subsequent two-qubit gate involving these qubits is ready to be transformed.
  2. Qubit Progress Tracking (PG). To make decisions the algorithm tracks how busy the each qubit is. They use a counter that tracks operations in the qubit.
  3. SABRE original score is used and a penalty term is added based on a PG score that are involved in SWAP. When the SWAP operator position is decided then it is calculated when it is the best time to add it.

  They They applied this extension to SABRE algorithm.

  Observations: By using this algorithm the circuit depth is reduced (according to 27 % on average depth is reduced), is quite scalable but it might result in more SWAP operators and it increases compilation time.

- Parallel qubit mapping problem [AV24]. Binary Integer Non-Linear Programming (BINLP) can be used to solve parallel qubit mapping problem [AV24]. Algorithm solution is divided into two parts:
  1. Clustering. Quantum computer topology is partitioned into clusters and each circuit is assigned to each cluster. Clusters are selected in that way that it estimates error cost of placing SWAP operators between any two-qubits in the potential cluster
  2. Fine-grained mapping is performed. It is decided which virtual qubit from quantum circuit maps to a physical qubit within its assigned cluster.

  Observations: it can be used to run multiple circuits at the same time, clustering approach can also be applied to single circuit optimisation by cutting circuit into sub-circuits.

- Another algorithm working with running algoritms in parallel - QuCloud+ [LD24]. Community Detection Assited Partitioning (CDAP) is used to find initial qubit mapping. The hierarchy tree of the physical qubits based on their reliability and connectivity is built. Community detection algorithm is used to cluster qubits with taking consideration about crosstalk. Within a chosen cluster, the logical qubits are mapped to the physical qubits prioritising ones with higher connectivity to reduce SWAP operators usage. In this algorithm SWAPS are also added between two clusters and this reduces SWAP operators count (they call it X-SWAP).

  Observations: While this algorithm is proposed for parallel run, this CDAP partitioning improves fidelity and reduces crosstalk. But X-SWAP part will not be used for single

circuit run.

- Reducing errors after qubit mapping [NT23]. Isomorphic graphs structure is also used in MAPOMATIC algorithm [NT23]. Its goal is to reduce errors after qubit mapping. Algorithm steps:
  1. The circuit is first compiled to have a circuit that is only consisting of basic gates and SWAP operators are already added where needed
  2. Then the algorithm analyses the compiled circuit and creates interaction graph.
  3. Then isomorphic subgraphs are searched where the goal is to find all possible physical qubit layouts that have the same structure as the quantum circuit's graph.
  4. Each layout is scored by using heuristic cost function. Total fidelity is calculated for the circuit if it were run on the specific layout.
  5. The layout with the lowest estimated error is selected.

  Observations: according to the paper it can recover around 40 % fidelity and it is used after compilation so it can be added to any algorithm. But the problem is that if the initial compilation was poor it will not remove SWAP operators it will only change to better performing qubits.

One of known alternatives to SWAP operators in routing part is to use Bridge operators. Interesting take on using Bridge operators with SWAP operators was seen in some research articles [CSS+25]. According to this article, algorithm performance difference with and without Bridge operators is negligible.

Some of proposed algorithms compare their performance with SABRE algorithm [LDX19b]. To see how they rank with each other and also to get an overview of the all algorithms that were analysed, results are presented in Table 3.

Table 3. Combined summary of qubit mapping algorithms, techniques, and performance. The algorithms are compared to the SABRE algorithm

Algorithm	**Core techniques**	**Fidelity improvement (%)**	**Gate reduction (%)**
QMBIP [JFD+25]	Binary Integer Programming, Circuit Slicing	53,9 (absolute)	-
Tabu Search [JDX24]	Tabu Search (Metaheuristic), Subgraph Matching	-	30
MQT QMAP [WB23]	Satisfiability Solving (Exact/Heuristic)	-	-

1D Trapped-ion QC [TCJ24]	Satisfiability Modulo Theories (SMT)	-	-
Optimal Subarchitectures [PBW23]	Sub-architecture Generation, Routing	-	-
QUBO [UL23]	Quadratic Unconstrained Binary Optimization	-	-
A doubly stochastic matrices [MZ23]	Continuous Optimization, Rolling Horizon	-	-
SWin [FZW+23]	Sliding Window, A* Search	-	~15
EffectiveQM [CSS+25]	Local Search (Exploitation/Exploration)	-	-
Fidelity-Optimal Qubit Mapping [SIM+23]	Brute-force Subgraph Search, Heuristics	25	10,52
AOQMAP [JKP24a]	Suzuki-Trotter Decomposition	-	43,4
SA-DQN [LLW+24]	Reinforcement Learning (Deep Q-Network)	-	-
GNAQC [TFX+23]	Reinforcement Learning (GNN, Q-Network)	12,7	-
PSO [BDS25]	Particle Swarm Optimization (Metaheuristic)	-	~16
QM-DLA [LZZ+24]	Dynamic Look-ahead, Heuristic Cost	-	23,95
Doustra [CYK+24]	Depth-First Search, Dijkstra's Algorithm	-	~23
Single-Qubits matter [LNC+23]	Algorithm Enhancement (SABRE extension)	-	-
QuCloud+ [LD24]	Community Detection for parallel execution	10,56	9,1
MAPOMATIC [NT23]	Post-processing, Isomorphic Error Reduction	-	-
PauliForest [YYH+24]	Pauli string mapping	-	-

2.5.2. Response to question 2

The question 2 was: *How to measure performance of algorithms that solve qubit mapping problem?*

Most common measurement techniques:

- Optimisation time [BDS25; CSS+25; CYK+24; FZW+23; JFD+25; LZZ+24; MZ23; NT23; PBW23; SIM+23; TCJ24]. The total time taken by an algorithm to transform quantum

scheme into one that is runnable on the selected quantum computer or more efficient. This is about classical computational speed. The less amount of time is better.

- Inserted two-qubit gate count. [CSS+25; CYK+24; JFD+25; JKP24a; LZZ+24; MZ23; SIM+23; YYH+24] The number of CNOT gates that were added after quantum circuit optimisation. The fewer the gates, the better. The lower count means lower noise.
- Depth [BDS25; FZW+23; JKP24a; LLW+24; LNC+23; MZ23; YYH+24]. The number of layers in the quantum circuit. It shows the minimum number of steps that is needed to execute the presented quantum algorithm. The lower amount the better.
- Fidelity [JFD+25; NT23; SIM+23; TFX+23]. Fidelity shows how accurately the implemented quantum circuit returns the ideal theoretical output. The circuit fidelity is calculated and is a product of the fidelities of all the gates. Gate fidelity is the closeness of an implemented gate to its ideal unitary action [SMM+23]. The higher, the better.
- Approximation ration [FZW+23]. This shows how far away the algorithm's solution is from optimal solution.
- Execution time of the circuit. According to the CODAR single-qubit and double qubit execution time is different [DZL20]. Due to this reason Cheng et al. scientists [CYK+24] use units of time to calculate mapping costs.
- The Discounter Quadratic Routing Length (DQRL) [UL23]. This metric asses the quality of given qubit assignment. An upper bound number of additional SWAP gates that is introduced by using a routing algorithm. This metric has evolved from Very Large Scale Integration (VLSI) technology where VLSI Placement and Routing problem exists [SM91]. Objective of that problem is to minimise the wire length that is used to connect components.

It can also be seen that in the most papers that had benchmarks compare their proposed solution with another set of algorithms. One of the most popular sets is QASM benchmark set [LSK+22].

2.5.3. Response to question 3

The question 3 was: *How can quantum optimisation algorithms be further optimised compared to ordinary quantum algorithms?*

Special algorithms can be created that are suited for variational quantum algorithms. Quantum optimisation algorithms typically are defined by a problem Hamiltonian [JKP24a; YYH+24]. Hamiltonian is composed from a specific set of Pauli strings. The proposed PauliForest algorithm [YYH+24] exploited this structure for finding initial qubit mapping (See in response to question 1). This allows to reduce the number of needed SWAP gates compared to general solutions. For QAOA, the structure of this algorithm can be further optimised by using quantum synthesis [JKP24b]. The AOQMAP algorithm constructs the most efficient implementation of a QAOA layer on the selected quantum computer rather just routing gates with SWAP operators by using predefined templates.

2.5.4. Response to question 4

The question 4 was: *How does the topology of a quantum computer influence the performance of quantum circuit optimisation algorithms?*

- For some algorithms they tend to work worse when the chips are high connectivity [FZW+23; JDX24]. Higher connectivity graphs have more edges implies bigger solution space. For some greedy algorithms it can take too much time to find a solution. But in the contrary, some algorithms do take advantage of denser topologies as it was written in the PauliForest study [YYH+24].
- Optimal solution algorithms may be unusable to the long calculation time for bigger quantum computer topologies [SIM+23]. These type of algorithms tend to iterate through all possible solutions without cutting corners. This can lead to exponential time which is unusable for qubits that have more than 15 qubits.
- Algorithms can also be created for specific architectures. One example is when the scientists applied qubit interaction graph on hexagonal qubit architecture [KDS+24]. By having a hexagonal structure, the number of SWAP gates can be reduced by taking advantage of the specific structure.
- Almost every research is working on superconducting quantum computers because the qubit mapping problem is the main problem for this type of computers. But there is a research article that worked on trapped-ion architecture quantum computers [TCJ24]. Qubits in that architecture are held in a line called channel. They can interact with each other by providing a full graph. But the unique constraint is that two sets of qubits cannot be operated on the same time if their physical locations overlap.

2.5.5. Response to question 5

The question 5 was: *What are the identified research gaps and unresolved challenges in current literature regarding qubit mapping and circuit optimisation?*

- Cost of current optimal solution is still too high because their proposed algorithm tries to pursue two goals in some scenarios [FZW+23].
- Adding other factors like quantum age execution error and quantum bit coherence time [LZZ+24].
- There is a need to reduce the crosstalk errors [CYK+24; LNC+23]. Crosstalk is identified as a major type of noise. This happens from unwanted qubit interactions when multiple quantum gates are executed concurrently [XZZ21]. Due to these errors there is a 20 times error rate increase for CNOT gates which can lead to incorrect output values.
- One of the new research topics is how to optimise execution of multiple circuits simultaneously on the same machine [AV24]. Quantum computers are often underutilised because users tend to run small circuits and many qubits are left unused.

2.6. Summarised view of the analysed qubit mapping algorithms

The table 4 presents a summarised view of the qubit mapping algorithms analysed in the literature review. The algorithms are categorised based on their underlying techniques, complexity, scalability, and key insights or trade-offs associated with each approach. Following scales were used to evaluate complexity and scalability:

- Complexity: 1 - Low, 2 - Low/Medium, 3 - Medium, 4 - High, 5 - Very High
- Scalability: 1 - Low, 2 - Low/Medium, 3 - Medium, 4 - High, 5 - Very High

Table 4. Comparative analysis of qubit mapping approaches based on criteria, complexity, and scalability.

Algorithm	**Technique**	**Complexity**	**Scalability**	**Insights & trade-offs**
Mathematical optimisation				
QMBIP [JFD+25]	Binary Integer Programming (BIP) with circuit slicing and triangle pattern decomposition.	4 - requires specialized solvers, computationally intensive.	2 - limited to sparse architectures.	Prioritizes high-fidelity connections. Slicing helps, but exact solutions are generally too slow for large modern devices.
MQT QMAP [WB23]	MaxSAT / SMT Solvers. Exact solution finding.	5 - NP-hard complexity.	1 - limited to ≈8 qubits.	Guarantees the optimal result but is effectively unusable for current and upcoming large-scale quantum computers due to runtime.
SMT-Ion [TCJ24]	SMT for time-step mapping. Divide and conquer.	4 - slower than heuristics, improved by divide & conquer.	3 - specific to Trapped-Ion (fully connected).	Excellent for fully connected topologies where the constraint is preventing gate overlap rather than routing paths.
Local search algorithms				
Tabu Search [JDX24]	Subgraph Isomorphism + Tabu Search with look-ahead.	2 - fast heuristic method.	4 - good scaling, but performance drops on dense graphs.	Fast execution. Sensitive to graph density (denser topology = less effective initial mapping). Risk of sub-optimal heuristic results.

Continued on next page

Table 4 – continued from previous page

Algorithm	Technique	Complexity	Scalability	Insights & trade-offs
EffectiveQM [CSS+25]	Two-mode search: Exploitation (Local) & Exploration (Global).	3 - slower than simple greedy methods due to mode switching.	4 - good for NP-hard problems.	Structured escape from local optima using exploration mode. Difficult to tune hyperparameters for specific topologies.
PSO [BDS25]	Particle Swarm Optimization (Global neighborhood heuristic).	1 - fast execution.	Uncertain. Not tested on big circuits.	Fast, but high probability of returning sub-optimal results on larger circuits due to probabilistic nature.
Lookahead strategies				
SWin [FZW+23]	Sliding Window + A* Search (Fixed-depth analysis).	3 - efficiently analyzes fixed-depth sections.	3 - sensitive to high connectivity.	Produces shallower circuits than basic greedy methods. High connectivity leads to search space explosion.
QM-DLA [LZZ+24]	Dynamic Multi-window look-ahead with heuristic cost.	4 - quite slow calculations.	3 - calculation speed is a bottleneck for big topologies.	Reduces SWAP count effectively (comparable to SABRE) but implementation is complex and slow.
Doustra [CYK+24]	DFS (Initial) + Dual-source Dijkstra (Routing).	1 - fast greedy approach.	5 - identified as ”best scaling” and ”future proof”.	Considers gate time and occupancy. Risk of getting stuck in local optima for highly structured or repetitive circuits.
Decomposition strategies				
Optimal Sub-arch. [PBW23]	Partitioning into optimal sub-architectures.	5 - optimization time acceptable only for < 30 qubits.	1 - too small for large scale research.	Returns optimal solutions but hits a computational wall quickly; not suitable for large devices.
Fidelity-Optimal [SIM+23]	Brute-force Subgraph Search + Heuristics.	5 - brute-forcing sub-topologies is inefficient.	1 - bottleneck for large topologies.	Reliable initial mapping, but the exhaustive search method makes it inefficient for large quantum computers.
Specialised for quantum optimisation				

Continued on next page

Table 4 – continued from previous page

Algorithm	Technique	Complexity	Scalability	Insights & trade-offs
AOQMAP [JKP24a]	Suzuki-Trotter Decomposition (QAOA specific).	3 - depends on classical optimization loop.	3 - dependent on pre-defined templates.	Highly effective for QAOA (43% gate reduction), but requires specific templates for new hardware.
PauliForest [YYH+24]	Pauli String feature matrix / Static mapping.	1 - static approach.	2 - poor on sparse graphs.	Excellent for Pauli-Z based optimization algorithms. Benefit decreases as hardware graph becomes sparser.
Machine learning				
SA-DQN [LLW+24]	Deep Q-Network (RL) + Simulated Annealing.	5 - requires training and tuning.	3 - unclear comparison to heuristics.	Combines optimization techniques, but training overhead and tuning complexity are significant barriers.
GNAQC [TFX+23]	Graph Neural Networks (GNN) + Q-Network.	5 - relies on accurate simulation for rewards.	3 - limited by simulator speed.	Improves fidelity (12,7%) via better initial mapping. Dependent on simulator accuracy; routing remains standard SABRE.
Novel continuous approaches				
SSWAP [MZ23]	Doubly Stochastic Matrices (Continuous Optimization).	5 - computationally intensive math.	1 - difficult to compute for large instances.	Reduces depth significantly using ”partial swaps”, but is mathematically heavy, hard to comprehend, and difficult to implement.

3. Complementary literature analysis

A systematic literature review of qubit mapping problem has identified some several potential enhancements to existing methods. To further investigate these enhancements, this systematic literature review was supplemented by a non-systematic analysis that was designed to gather specific novel implementation data.

3.1. Subgraph isomorphism problem

After qubit mapping existing solution analysis it can be seen that initial qubit mapping problem can be represented as a subgraph isomorphism problem. Unlike the full systematic review conducted for the qubit mapping problem, the investigation of the subgraph isomorphism algorithms serves a complementary role and is treated here as a targeted literature scan to identify potentially useful techniques for initial qubit mapping. To find solutions that were not used, articles from 2024 will be only included. Number of selected articles for this analysis is limited to four.

3.1.1. Quality characteristics

Three quality characteristics were set for the studies:

1. The article's central theme is about subgraph isomorphism problem.
2. The article compares, proposes or analyses algorithms that are used to solve subgraph isomorphism problem.
3. The article explicitly describes, proposes or analyses metrics, benchmarks that were used for evaluating the performance of subgraph isomorphism problem.

Selected studies are shown in Table 5.

Table 5. Analysed subgraph isomorphism problem articles

Database	Paper title	QC1	QC2	QC3	Sum
Web of Science	ENDNet: Extra-Node Decision Network for Subgraph Matching	1	1	1	3
Web of Science	A Generalized Community-Structure-Aware Optimization Framework for Efficient Subgraph Matching in Social Network Analysis	1	1	1	3

ACM	BOX: Subgraph Matching with Batch Backtracking Search	1	1	1	3
IEEE	Large Subgraph Matching: A Comprehensive and Efficient Approach for Heterogeneous Graphs	1	1	1	3

3.1.2. Possible algorithms

- Expanding graph neural networks (GNNs) by using an algorithm called ENDNet (extra-node decision network) [SAK25]. In this approach influence of extraneous nodes is detected and mitigated. This algorithm has improved accuracy from another trainable subgraph matching models from 91,6 % to 99,1 %. Steps of the algorithm:
  1. For each pair of nodes their cosine similarity is calculated on their current feature embeddings. This score is used to create predicted matching matrix M, where $M(i,j)$ represents the model's confidence that query node i matches data node j.
  2. For each node j in the the data graph, in the representing column in M the maximum matching probability is found with any node in the query graph. This probability is compared against learnable threshold β. If the probability is lower than threshold, the node j is set as extraneous. Extraneuos node feature vector is set to zero.
  3. Both query and data graphs are processed by the same GNN.

  To use this algorithm for initial qubit mapping problem, the query graph would be defined as a quantum circuit and the data graph would be the quantum computer topology.

- Generalised Community-structure-aware optimisation Framework (GCF) [LW24]. A core idea is to use inherent community structure to break problem into smaller parts and to prune search space. Steps of the algorithm:
  1. Partition data graph into communities. Community is a set of vertices that are densely connected internally but sparsely connected to vertices in another communities.
  2. Using c-schemes (community distribution schema). A c-scheme is a mapping that assigns each node of the small pattern graph to a community in a data graph. GCF generates all possible c-schemas and solves matching problem for them.
  3. Iterating all c-schemas would be inefficient. Three optimisation strategies are used:
     - Community path based pruning. For each c-schema it checks if connectivity requirements are fulfil for pattern graph. If pattern graph has an edge between two vertices it has to have an edge between two nodes in a community.
     - Community structure based Boundary Pruning. If the pattern vertex v_a (that is in the community $C1$) has an edge with v_b that is mapped in a different community $C2$,

then the data graph vertex matching v_a has to be a vertex in $C1$ that has a physical edge to a vertex in $C2$.
  - Two-level Symmetry-Breaking. If two c-schemes are equivalent due to the symmetry, the algorithm only needs to compute results for the one scheme.

  To use this algorithm to find initial qubit mapping, the patter graph would be the quantum circuit and the data graph would be the quantum computer topology graph.

- Subgraph Matching with Batch Backtracking Search. A search box perspective is used. Instead of a single path, space is viewed as a box that is a Cartesian product of the candidate data vertices for each query vertex. Then the recursive batch loop is used:
  1. Deciding which query vertex to focus. It is a dynamically selected one which is the most likely to accelerate the search.
  2. The batch of data vertices is selected. The group of data vertices is the vertices that are likely to produce similar or identical subsequent search boxes. This is done by clustering candidates that share high degree of internal similarity.
  3. Then box is refined, smaller new search box is created. And it is done until the box shrinks to the minimum.

  To use this algorithm for initial qubit mapping problem, the query graph would be defined as a quantum circuit and the data graph would be the quantum computer topology.

- Using Clustered Compressed Sparse Row and Sequential Candidate Equivalence algorithm [CWL+24]. All instances or embeddings of a small pattern graph are found in larger data graph with special focus for large, heterogeneous graphs. In this algorithm isomorphic edges are clustered. Algorithm steps:
  1. Data graph is scanned once and it groups all the edges into clusters. Edge and two vertices that connects it goes to one cluster.
  2. Online query processing. By using a pattern graph in this phase all embeddings are found. Efficient matching order is used:
     - Pattern edges are prioritised that have fewest potential matches in the data graph. The cluster's index is used to estimate this - a smaller cluster size means a higher constraint.
     - Then the dependency graph is built. if two parts of the pattern are connected via already-matched vertices, the search can be done independently.
     - By using that dependency graph it refines it matching order. It prioritises matching vertices that influence the largest number of subsequent vertices.
  3. Once the optimised matching order is determined, join-based execution engine is used to systematically build the embeddings one vertex at a time by following this order.

  To use this algorithm to find initial qubit mapping, the pattern graph would be the quantum circuit and the data graph would be the quantum computer topology graph.

Recent subgraph isomorphism algorithms address the qubit mapping problem in various ways

with machine learning and GNNs, community-based partitioning and optimised search techniques. The new GNN approach has increased its accuracy and the different GNN variant has been already used in some algorithms (GNAQC [TFX+23]). These algorithms represent promising candidates for improving initial qubit mapping search and are identified here as directions for future work. Their implementation was outside the scope of this thesis, which focuses on evaluating existing state-of-the-art mapping algorithms.

3.2. Quantum circuit synthesis

Quite similarly to subgraph isomorphism problem research, the quantum circuit synthesis analysis can also be useful when working with quantum optimisation algorithms. For this research, the depth analysis will not be conducted of the algorithm, only high level approaches what can be used to synthesise the unitary matrix to basic gates. Also, when using these algorithms, hardware constrains can be included in the the synthesis process which can result in better qubit placement. These algorithms will not be modified if they will be used. The number of studies was limited to three and studies from 2023 were included.

3.2.1. Quality characteristics

Three quality characteristics were set for the studies:

1. The article's central theme is about quantum circuit synthesis isomorphism problem.
2. The article compares, proposes or analyses algorithms that are used to solve quantum circuit synthesis problem.
3. The article explicitly describes, proposes or analyses metrics, benchmarks that were used for evaluating the performance of quantum circuit synthesis problem.

Selected studies are shown in Table 6.

Table 6. Analysed quantum synthesis articles

Database	Paper title	QC1	QC2	QC3	Sum
Web of Science	Depth-optimized quantum circuit synthesis for diagonal unitary operators with asymptotically optimal gate count	1	1	1	3
Web of Science	Quantum circuit synthesis with diffusion models	1	1	1	3
ACM	Synthetiq: Fast and Versatile Quantum Circuit Synthesis	1	1	1	3

3.2.2. Possible algorithms

Recent quantum synthesis proposed algorithms:

- By using generative AI models. Denoising Diffusion Models (DMs) can be used to generate quantum circuit [FMB24]. The main idea is to treat circuit synthesis like image generation. This approach works as follows:
  1. Model proposes a circuit.
  2. In the regular machine learning approach the proposed circuit is simulated to get the output.
  3. The output is compared to the desired target output using a cost function.
  4. The model is updated to minimise this cost

  In this proposed algorithm, simulating step is omitted that allows this synthesis to scale.

- Diagonal unitary operator is synthesised by using basic gate set of CNOT, Rz [ZHL24]. The high level algorithmic steps:
  1. Required Phase Shifts are calculated from the target diagonal operator.
  2. An empty circuit grid with n qubit rows and enough columns is created to allow calculations to run in parallel.
  3. Using a clever ordering rule the algoritm directly places all the necessary CNOT and Rz operators onto the grid.

  This approach declares that it reduces depth of specific QAOA algorithm circuits by 22,05 %.

- Synthetiq [PDB+24]. It uses simulated annealing to find a circuit. Algorithm steps:
  1. Synthetiq starts with a randomly generated circuit. The circuit's length is from a range that is updated based on the best solution so far.
  2. In each step the new candidate circuit is generated by making a random change to the current circuit.
  3. Domain-specific energy function is used to calculate if the proposed circuit is better.
  4. Process is repeated multiple times in parallel.

  Set of gates can be customised in this algorithm.

These proposed algorithms show that quantum synthesis is advancing by using different strategies like deterministic placement, heuristic searches and AI. The generative AI approach is particularly interesting to its ability to pass classical simulation that can be helpful for the future when the number of qubits in the quantum computer will grow. But all these methods are essential for translating the algorithms, like quantum optimisation ones, into efficient circuits.

4. Used qubit mapping techniques and algorithms

4.1. Rustiq

Rustiq [BM24] provides advanced synthesis techniques that are specialised for the Pauli-based quantum optimisation algorithms. By using their created heuristics, the scientists were able to provide an option for the users to synthesise their circuits in a way that would reduce the number of the entangling gates or the circuit depth. Their proposed techniques are greedy. To do the synthesis, the Pauli network definition is created which represents Clifford circuit C that is capable of diagonalizing a specific set of Pauli operators. These Pauli Networks allows to maintain current state of the Pauli strings in the table. To identify which gate should be applied next, the score function is used. For low gate count, it selects the single chunk that maximises the score. For low depth approach, the qubit interactions are modeled as a graph. For that graph the Max Weight Matching problem is solved. This identifies the optimal set of parallel gates to apply in a single layer.

4.2. QiskitAI

Qiskit has recently introduced a new AI-powered transpiler pass [DKM+25; KVP+25]. It uses machine learning techniques to predict the optimal transpilation strategy for a given quantum circuit and quantum computer. To predict the strategy, the patterns in the circuit structure and hardware constraints are analysed. According to the Qiskit documentation, this approach can be effective for large scale quantum circuits. The reinforcement learning model is used to predict the strategy. Instead of optimising the whole circuit, it breaks the circuit into clusters of sub-circuits (blocks). Each cluster is optimised separately. Once the block is collected, the block is provided to the AI that predicts the best transpilation strategy for that block according to the given hardware constraints. The AI model has been trained on a large dataset of quantum circuits and hardware topologies. If the newly build block is better than the old block according to the evaluation metrics, it is kept. Otherwise, the old block is restored. the process is repeated until the whole circuit is optimised.

4.3. LightSABRE

LightSABRE [ZTH+24] is an optimised variant of the SABRE algorithm (Section 1.3.6). Difference from the original SABRE:

- Simplified heuristic function that streamlines the cost calculation for evaluating each potential SWAP operation.
- Written in Rust (SABRE is in Python).
- ”Release valve” mechanism is used that allows to backtrack to a previous state and try a different path if algorithm fails.

It was decided to implement two variations of this algorithm:

- Decay heuristic. This heuristic scores possible set of SWAP gates by how much they reduce the distance between qubits in the current front layer of executable gates. Qubits that were recently swapped receive a penalty, making those paths look more expensive and pushing the search toward alternative routes. This prevents for routing through the same congested region of the chip.
- Lookahead heuristic. This heuristic scores candidate SWAPs using both the current front layer and the next layer of gates that would become executable immediately after. The score is a weighted sum of the distance reduction for both layers, with the next layer discounted by a factor of 0,5 by default. Using this technique, it prevents from choosing a SWAP that looks good now but creates a worse position for the gates that follow.

4.4. Doustra

From the algorithms identified in the literature review, Doustra [CYK+24] was selected for implementation. For a detailed description of the algorithm, its techniques, and integration approach, see Section 2.5.1 in the literature analysis results.

4.5. PauliForest

From the algorithms identified in the literature review, PauliForest [YYH+24] was selected for implementation as it is specifically optimised for quantum optimisation algorithms. For a detailed description of the algorithm, its techniques, and integration approach, see Section 2.5.1 in the literature analysis results.

5. Qubit mapping testing framework

For qubit mapping algorithms evaluation on set of quantum circuits and quantum computer topologies the testing framework has been developed. It allows to run different qubit mapping algorithms on set of quantum circuits and quantum computer topologies and collect results for further analysis. This helps to have consistent results between all the parameters. The framework description will be described in the following sections:

- General description
- Main components
- Qubit mapping algorithms integration
- Quantum circuits selection
- Quantum computer topologies selection and application
- Error mitigation techniques
- Evaluation metrics
- Results collection and analysis

Framework is open sourced and can be found on Github (https://github.com/-gabkeib/qumeld) [KP26].

5.1. General description

The general information about the framework is presented below.

5.1.1. Main functionality

The main functionality of the framework is to provide an easy way to run different qubit mapping algorithms on a set of quantum circuits and quantum computer topologies. The framework allows to collect results and analyse them. The main features of the framework are:

- Easy integration of new qubit mapping algorithms.
- Easy integration of new quantum circuits.
- Easy integration of new quantum computer topologies.
- Automatic results collection and analysis.
- Support for different evaluation metrics.

5.1.2. Programming language

Programming language: Python. The Python programming language has been chosen due to its extensive libraries for main quantum computing frameworks (like Qiskit [Ale19]) and data analysis (like Pandas [tea20]). Also, Python is widely used in the quantum computing research community, making it easier to integrate various qubit mapping algorithms. Negatives of using Python are its slower execution speed compared to compiled languages like Rust. To solve the performance issue, libraries that use C, Rust under the hood are preferred. For package

management the uv[2] package manager has been used. To ensure type safety and more clear code structure the ruff[3] linter has been used.

5.1.3. Modular design

The key aspect of the framework is the modular design. Because the main goal of this framework is to used by reseachers, this framework is designed in a way that allows easy integration of new qubit mapping algorithms, quantum circuits, and topologies. Each component (algorithm, circuit, topology) is encapsulated in its own module. This modularity allows researchers to easily add or modify components without affecting the overall framework. To achieve this, abstract base classes and interfaces have been defined for each component type. The description of each component type:

- Qubit mapping algorithms. For the qubit mapping algorithms two main methods have been defined: `run_circuit()` and `run_hamiltonian()`. The first method is used to run the qubit mapping algorithm on a generic quantum circuit. The second method is used to run the qubit mapping algorithm on a set of Pauli strings that represent a Hamiltonian. This method is needed because the main quantum algorithms analysed in this work are optimisation algorithms and they use Pauli strings representation for their Hamiltonians (like QAOA). For some algorithms only one of these methods is implemented (like PauliForest [YYH+24]).
- Quantum circuits. The quantum circuits are represented as a Qiskit QuantumCircuit object. The circuits are loaded from a predefined set of circuits stored in the framework. Each circuit module has a method `get_circuit()` that returns the QuantumCircuit object.
- Pauli strings. The Pauli strings are represented as a list of tuples, where each tuple contains a Pauli operator and its corresponding qubit index. Each Pauli string module has a method `get_pauli_strings()` that returns the list of Pauli strings.

[2] uv package manager:Github

[3] ruff linter: Github

5.2. High-level architecture

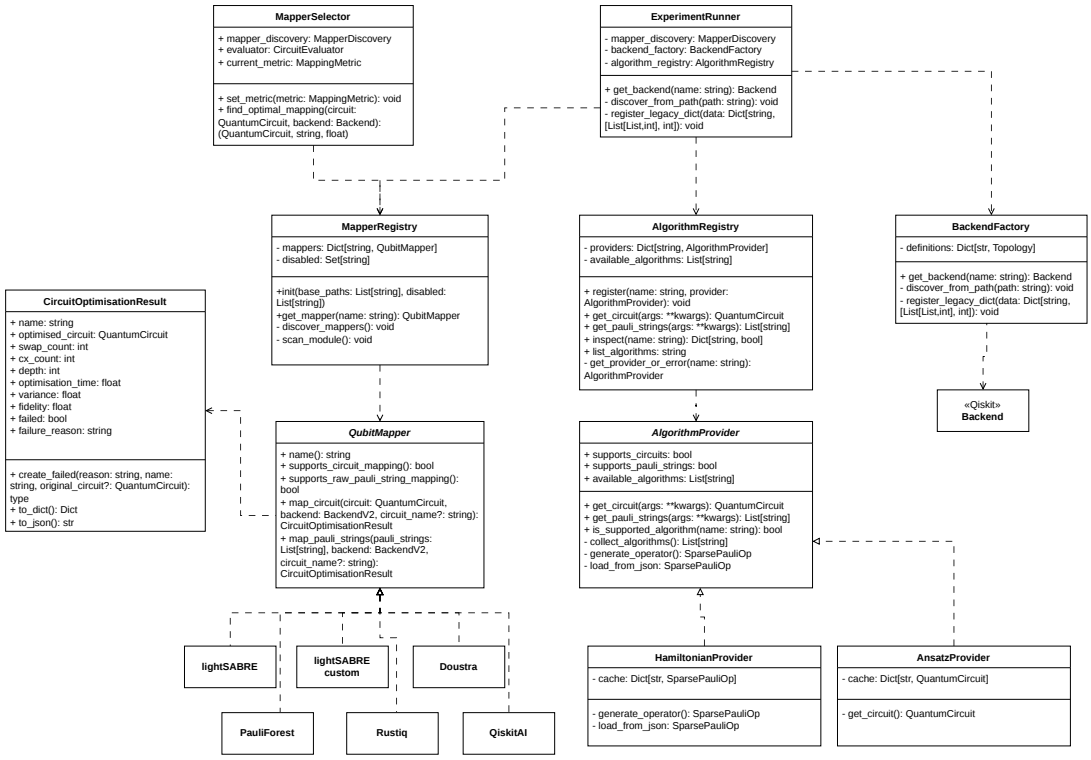

Figure 7. Class diagram of the framework architecture.

The main classes of the framework are presented in the figure 7. The main classes of the framework are:

- QubitMapper - represent interface for the qubit mapping algorithms. Each qubit mapping algorithm has to implement this interface.
- MapperRegistry - interface that is responsible for discovering and loading qubit mapping algorithms. It is the main source of the mapping algorithms that is used by other components. The list of the available mapping algorithms is discovered automatically by checking modules in the `mappers` package.
- MapperSelector - class that helps to find the best qubit mapping algorithm to use.
- CircuitOptimisationResult - class that represents the result of the circuit optimisation. It contains information about the mapped circuit, swap count, cx count, failure reasons and others.
- AlgorithmProvider - interface that is responsible for providing quantum circuits and Pauli strings for the qubit mapping algorithms to run on. Main use case of this is to load the circuits for the experiments.
- HamiltonianProvider - interface that is responsible for providing Pauli strings for the qubit mapping algorithms to run on. It uses automatic discovery to find the available Hamiltonians from the json files stored in the framework. Also the implementation of the QAOA Hamiltonian generator is present.

- AlgorithmRegistry - class that is responsible for registering and managing the qubit mapping algorithms. It uses the AlgorithmDiscovery load the algorithms.
- BackendFactory - class that is responsible for creating quantum computer backends based on the provided topology. It uses Qiskit BackendV2 class to create the backends and to be compatible with Qiskit infrastructure.
- ExperimentRunner - class that is responsible for running the experiments. It uses the AlgorithmDiscovery to get the circuits, BackendFactory to get the backends and MapperDiscovery to get the qubit mapping algorithms. It runs the experiments and collects the results.

The component diagram is presented in Figure 8. From this diagram it can be seen that it represents star architecture where `ExperimentRunner` acts as the core orchestrator for executing quantum computing experiments. It uses the `QubitMapper` to map circuits, `AlgorithmRegistry` is used to fetch the required circuits for testing. To test the optimal solutions, the `MapperSelector` is used. To get the topologies for testing the `BackendFactory` is used.

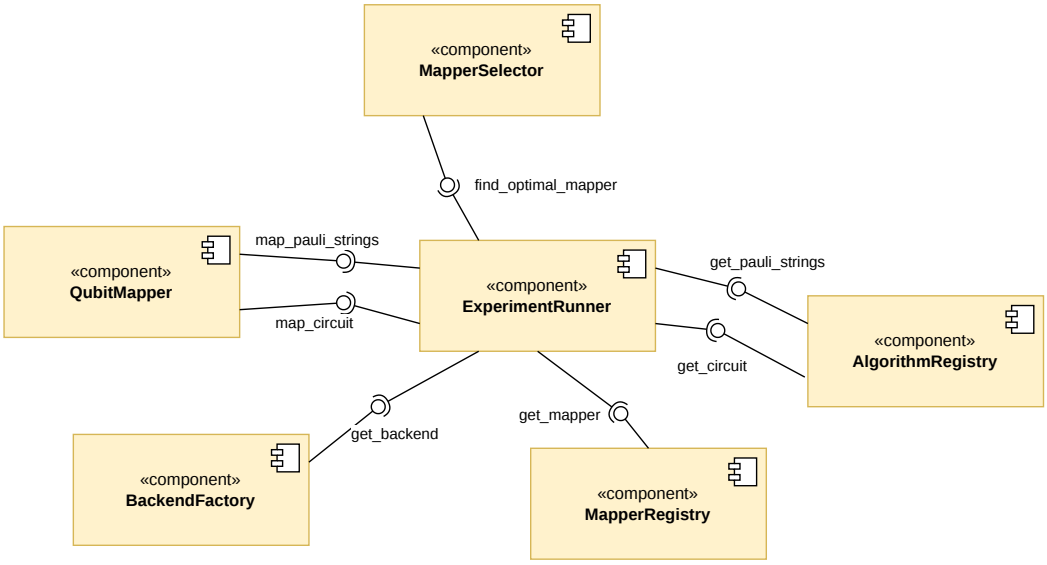

Figure 8. Component diagram of the framework.

5.3. Used design patterns

For the framework design, several design patterns have been used to ensure modularity, extensibility, and maintainability.

5.3.1. Factory pattern

The Factory pattern is used in the `BackendFactory` class. This class is responsible for creating quantum computer backends based on the provided topology. The factory pattern allows for the creation of different backend instances without exposing the instantiation logic to the client code. This makes it easy to add new topologies in the future and different sources of backends. One more reason to use this pattern for the backends is to build `BackendV2` instances from Qiskit

that are compatible with Qiskit transpiler and other components. This was needed to have a reasonable integration with Qiskit transpiler and other components.

5.3.2. Provider pattern

The Provider pattern is used in the `AlgorithmProvider` and `HamiltonianProvider` classes. These classes are responsible for providing quantum circuits and Pauli strings for the qubit mapping algorithms to run on. The provider pattern allows for the separation of the data source from the client code that uses the data. This makes it easy to add new circuits and Hamiltonians in the future without modifying the client code. The providers use automatic discovery to find the available circuits and Hamiltonians from the framework.

5.3.3. Template method pattern

The Template method pattern is used in the base `QubitMapper` interface and the `AlgorithmProvider` interfaces. The general workflow of circuit optimization (preprocessing, mapping, post-optimization) is defined in the abstract base mapping class, while specific implementations (LightSABRE, PauliForest, Doustra) override specific steps. This ensures consistent behavior across all mappers while allowing algorithm-specific customizations. The same is for the algorithm providers. This allows to have a consistent way of providing circuits and Hamiltonians while maintaining flexibility for different data sources.

5.4. Qubit mapping algorithms integration

Current state of the framework includes integration of the following qubit mapping algorithms:

- LightSABRE decay and lookahead heuristics [LDX19b].
- PauliForest [YYH+24].
- Doustra [CYK+24].
- Rustiq [BM24]
- QiskitAI [DKM+25; KVP+25]

5.4.1. LightSABRE integration

For the SABRE algorithm integration, it was decided to use two different implementations of the LightSABRE algorithm. Implementation differences:

- Default LightSABRE parameters. The default `PassManager` is created by using only the `generate_preset_pass_manager` method with optimisation level and backend topology provided. Optimisation level 3 has been used for the experiments to get the best results. By using this level, the largest number of layout and swap trials, max iterations is set.
- Custom LightSABRE parameters. The custom `PassManager` is created by providing two layers of the transpiler passes. For the layout transformation layer, the `SabreLayout` pass is used with the following parameters:

- `coupling_map=[graph]` - provided backend coupling map.
- `seed=static_seed` - random seed for the algorithm.
- `max_iterations=4` - the number of forward-backward routing cycles
- `layout_trials=200` - number of layout trials for the algorithm. Increasing the number can lead to better results, but increases the execution time.
- `swap_trails=200` - number of swap trials for the algorithm. Increasing the number can lead to better results, but increases the execution time.

For the routing layer, the `SabreSwap` pass is used with the following parameters:

- `coupling_map=[graph]` - provided backend coupling map.
- `heuristic="decay"` - heuristic for the algorithm. The decay heuristic is the one which dynamically weights qubits based on their probability of being swapped in the future. Another possible heuristic is ”lookahead”.
- `swap_trails=200` - number of swap trials for the algorithm. Increasing the number can lead to better results, but increases the execution time.
- `seed_transpiler=static_seed` - seed set for the algorithm.

For the backend topology, the `CustomBackend` class from Qiskit has been used to create a backend with the desired topology. This was further used as a class for the custom `ResearchBackend`. This included defining the coupling map and the basis gates. For the current implementation, the set of gates are same for the all topologies, only the coupling map and the possible connections between two-qubits gates are set according to the graph differs.

5.4.2. PauliForest integration

For the PauliForest algorithm integration, the PauliGo implementation from the PauliForest authors [YYH+24] has been used. The implementation is available on GitHub[4]. The source code has been copied into the framework and was integrated as a module. The main ideas behind this algorithm:

1. The Pauli strings are split into blocks based on their qubit interactions.
2. The most used qubit is placed at the center of the topology. This leads to minimization of the average distance between qubits.
3. The following qubits are placed based on their interaction with the already placed qubits. The more interactions a qubit has with the already placed qubits, the closer it is placed to them.
4. The process is repeated until all qubits are placed.
5. Then, by using distance matrix between qubits, the Pauli string blocks are mapped to the topology by using depth orientated scheduling.

Before running the transpile method, the set of Pauli strings have to be transformed to the set of Pauli blocks. To convert the Pauli strings to the set of Pauli blocks the following actions have been done:

[4]PauliGo implementation: Github

1. Pauli string is converted to the following format: `[[pauli_string, 1.0]]`, where `pauli_string` is a string representation of the Pauli string (like `'XZYIXZ'`). The second element in the list shows the rotation of the pauli operators.
2. This list of Pauli strings is converted to `pauliString` interface and the list of blocks is returned.

The main class is called `Compiler`. To create the quantum circuit, the compiler is initiated with the provided list of blocks and the quantum computer topology. Then, the compiler's start methods has to be called with the selected compilation method. For this framework, the `ucc` method has been used. The other possible methods are `ph`, `go`, `tk`. The ucc method is the most general one and is suitable for optimisation algorithms. After the compilation, the authors of the algorithm recommend to use the Qiskit transpiler with optimisation level 3 to further optimise the circuit. For this optimisation algorithm the generic circuit optimisation method is not supported - it only works with Pauli strings.

5.4.3. Doustra

The Doustra algorithm has been integrated by using the source code provided by the authors [CYK+24]. The implementation is available on GitHub[5]. The source code has been copied into the framework. The Doustra algorithm is a part of the Qsyn quantum circuit synthesis framework. It is written in the C++ programming language. To integrate it into the testing framework, it was decided to use Pythons subprocess module to call the C++ executable from Python. The decision to rewrite the algorithm in Python was discarded due to the complexity of the algorithm, to ensure that algorithms performance will be not affected and to save time. The Qsyn framework provides command-line interface (similar when calling `python` in the terminal), but also the scripts can be executed. The following approach is used to be able to integrate with the Python subprocess module. The main steps to run the Doustra algorithm are the following:

1. The quantum circuit is built. For the Pauli strings, the quantum circuit is built by using Qiskit `SparsePauliOp` class. The Pauli strings are added to the SparsePauliOp object and then the quantum circuit is built by using the `PauliEvolutionGate` class.
2. The parameter values for the Pauli operators are set. For the testing purposes, the random values are generated for each Pauli string.
3. The quantum circuit is exported to the OpenQASM format as the cache file (after receiving optimisation results the cache files are deleted). The Doustra algorithm accepts quantum circuits in the OpenQASM format.
4. The Doustra executable is called by calling the subprocess module. The quantum computers information is provided as a `.layout` files. These files contain the following information:
   - Quantum computer name.
   - Number of qubits.

[5]Doustra implementation: Github

- Set of gates supported by the quantum computer.
- Coupling map - connections between qubits.
- Error rates for each qubit and connection.
- Gate times for each gate.
- CNOT error rates for each connection.
- CNOT gate times for each connection.

The subprocess calls the Doustra executable with the following parameters:

```
doustra ./path/to/script.qsyn ./path/to/circuit.
    qasm ./path/to/out_circuit.qasm ./path/to/layout
    .layout
```

The Qsyn script is a file that contains the commands for the Qsyn framework. The script contains the following workflow:

(a) Load the quantum circuit from the OpenQASM file.
(b) Converts the circuit into zx-calculus representation. Runs optimisation passes on the zx-calculus representation.
(c) Converts the zx-calculus representation back to quantum circuit.
(d) Runs additional optimisations on the quantum circuit.
(e) Reads provided quantum computer layout file.
(f) Maps the quantum circuit to the provided quantum computer layout by using the Doustra algorithm.
(g) Exports the mapped quantum circuit to the output OpenQASM file.

5. After the subprocess execution is finished, the output OpenQASM file is read and converted back to the Qiskit QuantumCircuit object for further evaluation.

5.4.4. Rustiq

Rustiq algorithm is more orientated towards the quantum circuit synthesis rather than qubit mapping (which is also relevant for the optimisation algorithms qubit mapping) [BM24]. The Qiskit has created a Rustiq plugin that is designed for the low-depth compilation for the Hamiltonian simulation tasks. To use the Rustiq, the plugin is used as a high-level synthesis config parameter of the `PassManager` and is provided as the parameter to constructor. It is used as a PauliEvolution gate synthesis method. The following parameters are set for the Rustiq synthesis:

- `nshuffles=400` - number of shuffles for the greedy synthesis algorithm.
- `upto_phase=True` - whether to ignore global phase differences. This means that the returned circuit may differ from the original circuit by a global phase but it is typically not observed.
- `fix_clifford` - whether to fix the Clifford part of the synthesis.
- `metric="depth"` - metric to optimise for. Possible values are ”depth” and ”gate_count”.

To optimise the quantum circuit, the `PassManager` method `run()` is called.

5.4.5. Qiskit AI

The Qiskit AI-powered transpiler pass has been integrated by using the AI passes provided by Qiskit [DKM+25; KVP+25]. The passes are provided to the PassManager constructor as the `passes` parameter. The possible AI passes are:

- `AIRouting` - AI-powered routing pass.
- `CollectLinearFunctions` and `AILinearFunctionSynthesis` - linear function collection and synthesis pass.
- `CollectPauliNetworks AIPauliNetworkSynthesis` - Pauli network collection and synthesis pass.

Only `AIRouting` pass was used with parameters:

- backend - provided quantum computer backend.
- `optimization_level=3` - optimisation level for the transpiler.
- `layout_method="optimize"` - layout method for the transpiler.
- `local_mode=True` - whether to use local mode for the AI model.

It was decided to use only the routing pass because the other passes take a significant amount of time to execute and the main goal of this work is to evaluate the qubit mapping algorithms.

Not only it has performance issues, the current implementation of the transpiler pass has the limitation that it only supports library `networkx==2.8.5`. For the framework, the latest version of the networkx library is used. To solve this issue, a virtual environment has been created with the required version of the networkx library. The Qiskit AI-powered transpiler pass is executed in this separate virtual environment by using the Python subprocess module.. The subprocess calls the Python script that runs the Qiskit AI-powered transpiler pass with the following parameters:

```
python ai_transpiler.py run_qiskit_ai_circuit path/to/temp/
    file
```

The temporary file contains the `CircuitOptimisationRequest` object serialized as a pickle file. After the subprocess execution is finished, the output temporary file is read and converted back to the `CircuitOptimisationResult` object for further evaluation.

5.5. Quantum circuits selection

The circuits used for testing are defined by providing their Hamiltonian as a set of Pauli strings. The circuits are generated by using the qubit mapping framework itself. The following quantum algorithms are used for testing:

- QAOA Max-Cut problem Hamiltonian. The Max-Cut problem Hamiltonian is created by analysing the provided graph. If the there is an edge p_1, p_3 between nodes p_1 and p_3, then the Pauli string would be represented like this: $IZIZI$ for 5 nodes graph.
- QAOA Max-Cut problem ansatz. The Max-Cut problem ansatz are generated by using the

Qiskit `QAOAAnsatz` class. The simple graph as a matrix is provided and the `QAOAAnsatz` class generates the quantum circuit for the Max-Cut problem.

- VQE H_2 molecule circuit. The set of Pauli strings for the H_2 molecule is taken from the Qiskit examples. The Pauli strings with their coefficients are imported from the json file.
- VQE LiH molecule circuit. The set of Pauli strings for the LiH molecule is taken from the PauliForest examples. The Pauli strings with their coefficients are imported from the json file.
- VQE with random Hamiltonian. Predefined set of random Pauli strings is provided to generate the quantum circuit. The Pauli strings with their coefficients are imported from the json file.
- VQE used `EfficientSU2` ansatz. It is built by providing number of qubits in the Hamiltonian, number of repetitions and the selected entanglement type. For the testing purposes linear entanglement is used.

5.6. Quantum computer topologies selection and application

5.6.1. Supported topologies

The quantum computer topologies are defined as graphs. For each quantum computer topology, the correspoding function that returns the coupling map and the number of qubits is defined. The following quantum computer topologies are used for testing:

- Hexagonal lattice topology. The hexagonal lattice topology is generated by using a function that creates a hexagonal grid of qubits. Each qubit is connected to its adjacent qubits in the hexagonal pattern. 54 qubits version of this topology is used for testing.
- Google Willow topology. The Google Willow topology is defined by using the coupling map provided by Google [AAA+24]. 105 qubits version of this topology is used for testing.
- IBM Almaden - 20 qubits topology.
- IBM Cambridge - 28 qubits topology.
- IBM Eagle - 127 qubits topology.
- IBM Falcon - 27 qubits topology.
- IBM Heron - 133 qubits topology.
- IBM Manhattan - 65 qubits topology.
- IBM Montreal - 27 qubits topology.
- IBM Poughkeepsie - 20 qubits topology.
- IBM Reuschlikon - 16 qubits topology.
- IBM Rochester - 53 qubits topology.
- IBM Tokyo - 20 qubits topology.
- IonQ Harmony topology. The IonQ Harmony topology is a fully connected topology where each qubit can interact with every other qubit. 9 qubits version of this topology is used for testing.

- Rigetti Novera. The square lattice topology is used for testing. 9 qubits version of this topology is used for testing.
- Riken Fujitsu topology - 256 qubits topology.

5.6.2. Applying the topologies

To use presented quantum computer topologies in the framework, the `CustomBackend` class from Qiskit has been used to create a backend with the desired topology. This backend is further used as a class for the custom `ResearchBackend`. This included defining the coupling map and the basis gates. For the current implementation, the set of gates are same for the all topologies, only the coupling map and the possible connections between two-qubits gates are set according to the graph differs.

The characteristics of `ResearchBackend` class are the following:

- Each qubit $i \in \{0, 1, \ldots, n-1\}$ has randomly generated coherence and frequency characteristics:
  - T1 relaxation time: $T_1 \sim \mathcal{U}(50, 100)$ microseconds
  - T2 dephasing time: $T_2 \sim \mathcal{U}(20, 80)$ microseconds
  - Qubit frequency: $f \sim \mathcal{U}(4.5, 5.5)$ GHz
- Single qubit gates characteristics are defined in the 7 table.
- Two-qubit gates are defined on the coupling graph specified by the `graph_list` parameter:
  - CNOT (CX): Available on all edges (i,j) in the coupling graph with no specified error properties
  - CZ gate: Available on all edges with randomized characteristics:

$$\text{Error rate} \sim \mathcal{U}(7 \times 10^{-4}, 5 \times 10^{-3})$$
$$\text{Duration} \sim \mathcal{U}(10, 900) \text{ ns}$$

  - **SWAP:** Available on all edges with no specified error properties

Table 7. Gate error rates and durations for the `ResearchBackend` class.

Gate	Error rate	Duration
$R_z(\theta)$	0	0
X	$\mathcal{U}(10^{-6}, 10^{-4})$	$\mathcal{U}(10, 900)$ ns
$\sqrt{X}$	$\mathcal{U}(10^{-6}, 10^{-4})$	$\mathcal{U}(10, 900)$ ns
Measurement	$\mathcal{U}(10^{-3}, 10^{-1})$	$\mathcal{U}(10, 900)$ ns

5.7. Error mitigation techniques

For error mitigation techniques testing, the measurement error mitigation provided by Qiskit has been used. The measurement error mitigation is applied after the qubit mapping algorithm has

been applied and the quantum circuit is transpiled for the selected quantum computer topology. The following techniques are used:

- Dynamic decoupling. The dynamic decoupling is applied by using the `DynamicalDecoupling` pass from Qiskit.
- Pauli twirling. The Pauli twirling is applied by using the `PauliTwirling` pass from Qiskit.
- Twirled readout error extinction (T-REx). The T-REx is applied by using the `resilience.measure mitigation` and resilience.measure_noise_learning options from Qiskit.
- Zero noise extrapolation. The zero noise extrapolation is applied by using the `resilience.noise_amplification` option from Qiskit and by using Mitiq `zne` class.

5.8. Evaluation metrics

The evaluation metrics used to assess the performance of the qubit mapping algorithms are the following:

- SWAP count. The SWAP count is the number of SWAP gates added to the quantum circuit by the qubit mapping algorithm. A lower SWAP count indicates a more efficient mapping.
- Circuit depth. The circuit depth is the number of layers of gates in the quantum circuit after the qubit mapping algorithm has been applied. A lower circuit depth indicates a more efficient mapping.
- Expected value. The expected value is calculated for the smaller quantum circuits where the exact solution can be calculated by using the following formula:

$$\langle A \rangle = \langle \psi | A | \psi \rangle \,, \tag{1}$$

  where A is the observable (Hamiltonian) and ψ is the quantum state after the qubit mapping algorithm has been applied. The expected value is compared to the exact solution to assess the accuracy of the qubit mapping algorithm.
- Dispersion of the expected value. The dispersion is calculated by using the following formula:

$$\Delta(A) = \langle A^2 \rangle - \langle A \rangle^2 \,, \tag{2}$$

  where A is the observable (Hamiltonian) and ψ is the quantum state after the qubit mapping algorithm has been applied. The dispersion is used to assess the stability of the qubit mapping algorithm.
- Fidelity. The fidelity is calculated by using the following formula:

$$F(\rho, \sigma) = \left(\mathrm{Tr} \sqrt{\sqrt{\rho} \sigma \sqrt{\rho}} \right)^2 \,, \tag{3}$$

  where ρ is the density matrix of the quantum state after the qubit mapping algorithm has

been applied and σ is the density matrix of the ideal quantum state. The fidelity is used to assess the accuracy of the qubit mapping algorithm.

- Optimisation time (s) – the time taken by the qubit mapping algorithm to map the quantum circuit to the quantum computer topology. A lower optimisation time indicates a more efficient algorithm.

5.9. Results collection and analysis

The results from the qubit mapping algorithms are collected and stored under the `results/` directory. Each test run has its own subdirectory named with the timestamp of the run. Inside each subdirectory, the following structure is used:

1. First layer: the name of the quantum computer topology.
   (a) Second layer: the name of the tested quantum circuit (VQE, QAOA and others).
      i. Third layer: the json file that is named of the qubit mapping algorithm used (LightSABRE, PauliForest, Doustra).

The json file contains the following information:

- Quantum circuit tested.
- Swap count.
- Circuit depth.
- Expected value.
- Dispersion.
- Fidelity.
- Optimisation time (s).

These results can be further analysed by using the Jupyter notebook that is provided in the framework under the `analysis/` directory. The notebook allows to load the results from the `results/` directory (the most recent one is selected automatically). For the each quantum circuit algorithm the table comparing the qubit mapping algorithms is generated. The tables can be further exported to CSV files for further analysis.

5.10. Key workflows

The main workflows of the framework are the following:

- Optimising a quantum circuit with a qubit mapping algorithm on the specified quantum computer topology.
- Running experiments.

5.10.1. Optimising a quantum circuit

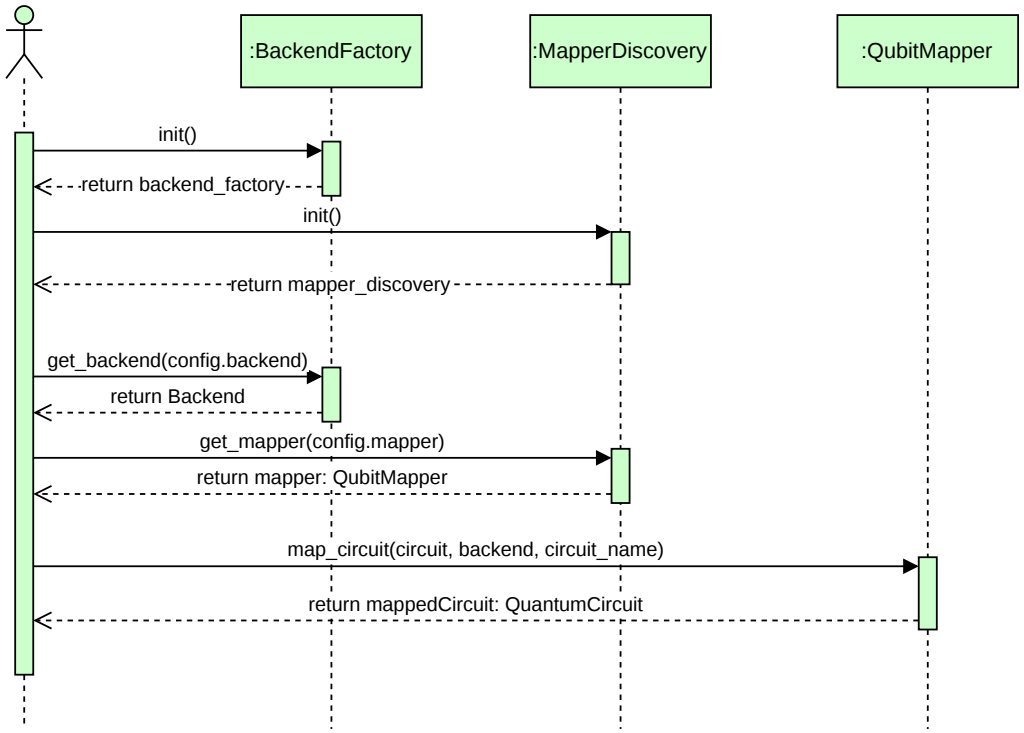

Figure 9. Sequence diagram of optimising a quantum circuit with a selected qubit mapping algorithm on the specified quantum computer topology.

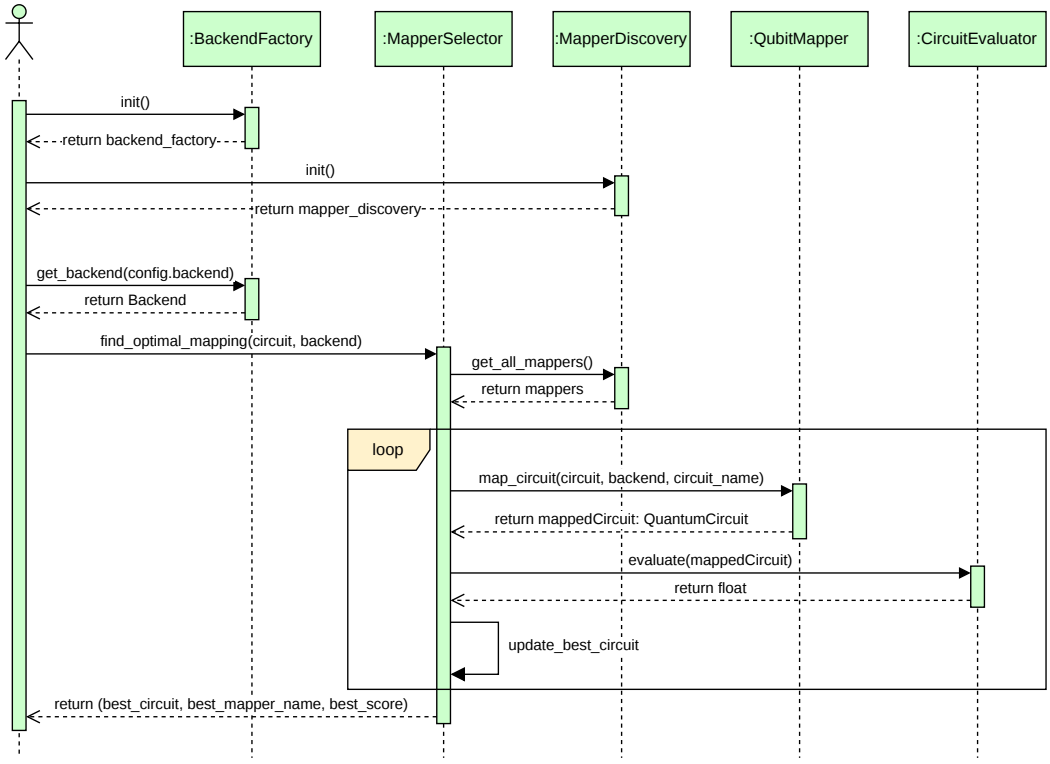

Figure 10. Sequence diagram of optimising a quantum circuit with an automatically selected qubit mapping algorithm on the specified quantum computer topology.

The two different approaches for optimising a quantum circuit can be used:

- By explicitly setting the desired qubit mapping algorithm. The sequence diagram of this workflow is presented in the figure 9. The user initialises the `BackendFactory`, `MapperProvider` classes. Then, by using the `BackendFactory` the quantum computer backend is returned based on the provided topology. The qubit mapping algorithm is retrieved from the `MapperProvider` by using the provided algorithm name. Then, the `map_circuit()` method is called on the qubit mapping algorithm with the quantum circuit. The result of the optimisation is returned as a `CircuitOptimisationResult` object.
- By running the `MapperSelector` that finds the most optimal qubit mapping algorithm for the given quantum circuit and quantum computer topology. The sequence diagram of this workflow is presented in the figure 10. The user initialises the `BackendFactory`, `MapperProvider`, `MapperSelector` classes. Then, by using the `BackendFactory` the quantum computer backend is returned based on the provided topology. The `MapperSelector` uses the `MapperProvider` to get the list of available qubit mapping algorithms. Then, for each qubit mapping algorithm the `map_circuit()` method is called with the quantum circuit. The results are collected and the best one is returned as a `CircuitOptimisationResult` object.

5.10.2. Running the experiments

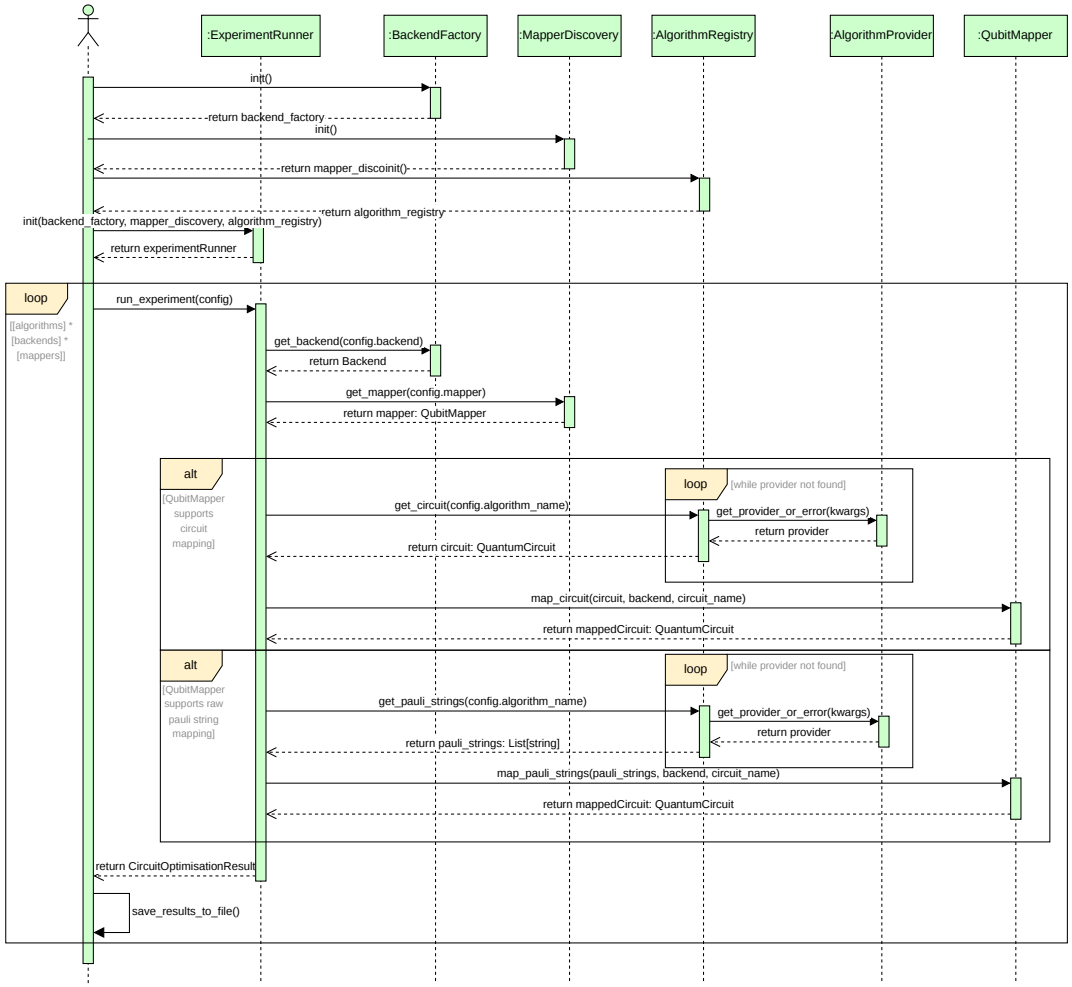

Figure 11. Sequence diagram of running the experiments.

In the 11 the sequence diagram of running the experiments is presented. The user initialises the `ExperimentsRunner` class with the `AlgorithmRegistry`, `BackendFactory`, `MapperDiscovery`. Before running the experiments users creates config object. Then, for the each experiment the `run_experiments(config)` method is called. The `ExperimentsRunner` uses the `BackendFactory` to get the quantum computer backend based on the name provided in the config. Then by using the `MapperProvider` instance the mapper that is defined in the config is retrieved. The quantum circuit is retrieved from the `AlgorithmRegistry` by using the name provided in the same config. Inside the AlgorithmRegistry, the list of the `AlgorithmProviders` is checked to find the required circuit or the set of PauliStrings (depends on the optimisation algorithm used). Then, the `map_circuit(config.algorithm_name)` method or the `map_pauli_strings(config.algorithm_name)` is called on the qubit mapping algorithm with the quantum circuit. The results are collected and stored in the `results/` directory.

6. Setup

6.1. Running environment

The experiments were run on a machine with the following specifications:

- Processor: M1 Pro Apple Silicon (8-core CPU, 16-core GPU)
- RAM: 16 GB
- Operating System: macOS Sequoia 15.3
- Python version: 3.13.7
- Qiskit version: 2.1.1

6.2. Experiments settings

- Six quantum circuits were selected to test:
  - VQE H_2 Hamiltonian
  - VQE LiH Hamiltonian
  - VQE random demo Hamiltonian
  - VQE ansatz (EfficientSU2)
  - QAOA Max Cut Hamiltonian
  - QAOA ansatz with Max Cut Hamiltonian
- Each set of algorithms was tested with every predefined quantum computer topology and qubit mapping algorithm (6 circuits $\times$ 16 topologies $\times$ 6 qubit mapping algorithms)
- The experiment execution was repeated for 15 times. For each column value that is presented in the tables, the median value was used. The median value was selected because it reflect closely to the expected result after qubit mapping execution.

6.3. Additional parameters for the algorithms

6.3.1. QAOA Max Cut graph

For the testing purposes it was decided to use small (5 qubits) graph for the analysis. The graph adjacency matrix:

$$G = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{4}$$

7. Experiments

In the following subsections the results for each quantum circuit will be presented.

7.1. Experiments results

For the concrete algorithm analysis, the two tables will be presented:

- Full results table - contains all the metrics collected during the testing. Will be presented in the appendix.
- Summary results table - contains set of quantum computers that differ in their topology type and number of qubits.

For the PauliForest algorithm, only results for Hamiltonians will be presented because this algorithm only works with Pauli strings representation of the quantum circuits. For the other tested circuits, PauliForest results will be marked as -1 in the results tables. The results for tested optimisation algorithms circuits that have more qubits than the topology will also have -1 value in the table.

7.2. VQE Hamiltonian with H_2 molecule

Table 8. Results for VQE H_2 Hamiltonian

Topology	Optimizer	SWAP	CNOT	Depth	Time (ms)
Google Willow Q105	doustra	30	126	65	77,77
	lightSABRE_decay	3	37	85	33,31
	lightSABRE_lookahead	3	35	85	98,52
	pauliforest	6	40	38	450,20
	qiskit_ai	8	60	192	104,08
	rustiq	3	39	87	32,77
Ibm Eagle Q127	doustra	18	79	67	62,76
	lightSABRE_decay	5	51	76	21,54
	lightSABRE_lookahead	5	49	71	141,78
	pauliforest	6	40	38	765,43
	qiskit_ai	6	54	221	108,51
	rustiq	5	51	76	20,10
Rigetti Novera Q9	doustra	16	73	58	45,49
	lightSABRE_decay	1	31	101	35,19
	lightSABRE_lookahead	1	34	100	90,61
	pauliforest	6	40	38	19,74
	qiskit_ai	7	57	208	102,67
	rustiq	3	37	93	27,79
Riken Fujitsu Q256	doustra	18	77	61	99,14
	lightSABRE_decay	3	41	111	48,56
	lightSABRE_lookahead	3	39	106	121,30
	pauliforest	6	40	38	6058,29
	qiskit_ai	6	54	225	152,21
	rustiq	3	39	86	48,95

Full results table for the VQE with H_2 Hamiltonian are presented in Table 27. The subset of topologies results are presented in the the table 8. From these results it can be seen that with the increasing qubit count in quantum computer the PauliForest algorithm takes the longest time to perform the qubit mapping (with circuit synthesis). It is 60 times slower than the second slowest one in regard with the biggest topology of all tested ones. But it has the lowest circuit depth. This can be explained by the fact that this algorithm synthesize the circuit from the given Pauli strings in its own way rather than with others where `SparsePauliOp` from Qiskit is used to generate the quantum circuit. Doustra algorithm produces highest amount of SWAP operators by having 82 - 91 % more. This leads to the fact that it also provides the biggest number of CNOT operators. For the lowest circuit depth the PauliForest algorithm is performing best on the heavy hex topologies (with the exception of IBM Manhattan, IBM Reuschlikon). For square and 2D lattice topologies the LightSABRE with lookahead heuristic produces the lowest CNOT count (10 % less than the PauliForest algorithm). The Qiskit AI-powered transpiler produces the biggest circuit depth for all topologies.

7.3. VQE Hamiltonian with random set of Pauli strings

Table 9. Results for random Pauli strings hamiltonian

Topology	Optimizer	SWAP	CNOT	Depth	Time (ms)
Google Willow Q105	doustra	175	590	190	702,53
	lightSABRE_decay	6	74	127	34,20
	lightSABRE_lookahead	6	74	128	221,42
	pauliforest	11	78	46	468,28
	qiskit_ai	40	180	316	141,43
	rustiq	7	77	127	40,20
Ibm Eagle Q127	doustra	184	615	228	250,19
	lightSABRE_decay	12	87	181	40,39
	lightSABRE_lookahead	12	87	177	276,67
	pauliforest	31	138	62	789,47
	qiskit_ai	27	141	451	148,02
	rustiq	12	87	177	40,13
Rigetti Novera Q9	doustra	77	290	151	206,06
	lightSABRE_decay	8	81	138	21,97
	lightSABRE_lookahead	8	81	138	165,32
	pauliforest	10	75	46	34,79
	qiskit_ai	33	159	335	129,89
	rustiq	8	81	137	19,60
Riken Fujitsu Q256	doustra	137	460	173	385,36
	lightSABRE_decay	8	77	169	65,75
	lightSABRE_lookahead	6	70	155	275,47
	pauliforest	11	78	47	6040,20
	qiskit_ai	25	135	451	179,00
	rustiq	7	74	168	81,00

Full results table for the VQE with random set of Pauli strings are presented in Table 28. The subset of topologies results are presented in the the Table 9. For the heavy hex toplogies

that are under 30 qubits, the Doustra algorithm takes the longest time to optimised (16,7 - 240 % longer). The biggest CNOT gates count is produced by the Doustra algorithm because it uses the most amount of SWAP operators (in some cases more than twice from the one in the second place with SWAP count). The lowest amount of CNOT gates and the circuit depth is produced by the LightSABRE with lookahead heuristic algorithm for all topologies except IonQ and Rigetti. For the last two mentioned, the PauliForest algorithm produces the lower CNOT count (7,4 - 21,1 % less) The PauliForest algorithm produces 70 % lower circuit depth compared to the other algorithms. Also, the differences between used SWAP strategies can be more for this algorithm. For example, for the Fujitsu Riken topology the LightSABRE with lookahead heuristic used 6 SWAP gates while the Doustra used 137 SWAP gates.

7.4. VQE Hamiltonian with LiH molecule

Table 10. Results for VQE LiH Hamiltonian algorithm

Topology	Optimizer	SWAP	CNOT	Depth	Time (ms)
Google Willow Q105	doustra	3277	11373	4756	9903,85
	lightSABRE_decay	1037	8242	14379	1309,12
	lightSABRE_lookahead	968	8026	13853	19659,53
	pauliforest	1635	7247	3334	1564,57
	qiskit_ai	3921	17824	32004	6422,29
	rustiq	1016	8119	14533	1389,97
Ibm Eagle Q127	doustra	5293	17435	6552	11050,04
	lightSABRE_decay	1987	10985	17872	1727,02
	lightSABRE_lookahead	1942	10849	17552	28680,15
	pauliforest	3087	11604	3988	2072,82
	qiskit_ai	6481	25503	33628	7592,73
	rustiq	2026	11158	14831	1562,50
Rigetti Novera Q9	doustra	-1	-1	-1	0,00
	lightSABRE_decay	-1	-1	-1	0,00
	lightSABRE_lookahead	-1	-1	-1	0,00
	pauliforest	-1	-1	-1	0,00
	qiskit_ai	-1	-1	-1	0,00
	rustiq	-1	-1	-1	0,00
Riken Fujitsu Q256	doustra	3330	11539	4728	13066,18
	lightSABRE_decay	994	8136	14362	1324,06
	lightSABRE_lookahead	953	7922	14940	20665,03
	pauliforest	1641	7262	3282	7116,02
	qiskit_ai	6417	25311	33853	7509,48
	rustiq	1027	8145	14576	1474,06

Full results table for the VQE with LiH molecule are presented in Table 29. The subset of topologies results are presented in the the Table 10. This was the most intense test of all circuits because it uses 12 qubits and 630 Pauli strings. For that reason, the IonQ Harmony and Rigetti Novera tests are marked with -1 values. The LightSABRE lookahead variant took the longest time to perform the qubit mapping, reaching 58,1 – 226,7 % longer than the next slowest method depending on topology. For CNOT count, PauliForest is best on Google Willow,

Hexagonal Lattice, IBM Almaden, IBM Tokyo, and Riken Fujitsu, improving the CNOT count by 6,8 - 9,7 % compared to the next best method, while LightSABRE lookahead is best on the remaining heavy-hex devices with a smaller margin of 0, 2- 5,8 %. For circuit depth, PauliForest is the best-performing algorithm on all executable topologies, reducing depth by about 21,4 - 78,6 % compared to the next best option.

7.5. QAOA Hamiltonian with the Max Cut problem

Table 11. Results for QAOA Max Cut Hamiltonian algorithm

Topology	Optimizer	SWAP	CNOT	Depth	Time (ms)
Google Willow Q105	doustra	24	97	50	50,03
	lightSABRE_decay	1	11	33	28,46
	lightSABRE_lookahead	1	11	33	50,49
	pauliforest	6	38	31	447,23
	qiskit_ai	3	21	41	91,23
	rustiq	1	9	42	26,33
Ibm Eagle Q127	doustra	17	75	56	46,03
	lightSABRE_decay	2	16	29	18,00
	lightSABRE_lookahead	2	16	29	43,06
	pauliforest	9	47	31	762,45
	qiskit_ai	4	24	17	93,16
	rustiq	2	18	25	17,17
Rigetti Novera Q9	doustra	18	82	52	35,31
	lightSABRE_decay	1	11	34	16,28
	lightSABRE_lookahead	1	11	33	33,33
	pauliforest	6	38	31	15,42
	qiskit_ai	2	18	32	86,04
	rustiq	1	11	42	15,39
Riken Fujitsu Q256	doustra	31	123	67	75,38
	lightSABRE_decay	1	11	43	53,37
	lightSABRE_lookahead	1	11	44	80,37
	pauliforest	6	38	31	6084,21
	qiskit_ai	4	24	18	129,15
	rustiq	1	9	39	55,51

Full results table for the QAOA with the Max Cut problem are presented in Table 30. The subset of topologies results are presented in the the Table 11. For this problem it can be seen tha the Doustra algotihm uses the highest amount of SWAP operators and CNOT gates. For the lowest amount of CNOT gates the both heuristics of the LightSABRE and Rustiq algorithms produces the best results for the majority of topologies (~15 % less). For the IBM Heron the Rustiq algorithm has produced the lowest amount of CNOT gates. For the lowest circuit depth the Qiskit AI-powered transpiler produces the best results for the majority of topologies (~45 % less than the second best). For the Rigetti Novera, IBM Reuschlikon and Google Willow topologoes the PauliForest algorithm produces the lowest circuit depth.

7.6. VQE ansatz (EfficientSU2)

Table 12. Results for VQE EfficientSU2 ansatz

Topology	Optimizer	SWAP	CNOT	Depth	Time (ms)
Google Willow Q105	doustra	21	85	61	618,87
	lightSABRE_decay	0	2	22	32,47
	lightSABRE_lookahead	0	3	15	33,62
	pauliforest	-1	-1	-1	0,00
	qiskit_ai	3	13	30	93,43
	rustiq	0	2	23	31,33
Ibm Eagle Q127	doustra	16	64	54	480,60
	lightSABRE_decay	0	4	16	12,69
	lightSABRE_lookahead	0	4	12	31,17
	pauliforest	-1	-1	-1	0,00
	qiskit_ai	0	4	16	90,62
	rustiq	0	3	13	11,49
Rigetti Novera Q9	doustra	16	67	47	178,10
	lightSABRE_decay	0	4	12	7,80
	lightSABRE_lookahead	0	3	22	22,69
	pauliforest	-1	-1	-1	0,00
	qiskit_ai	2	10	29	83,01
	rustiq	0	4	12	6,57
Riken Fujitsu Q256	doustra	16	64	54	825,76
	lightSABRE_decay	0	3	17	49,27
	lightSABRE_lookahead	0	2	21	93,78
	pauliforest	-1	-1	-1	0,00
	qiskit_ai	0	4	16	120,78
	rustiq	0	3	17	48,43

Full table results for the VQE ansatz (EfficientSU2) are presented in Table 31. The subset of topologies results are presented in the the Table 12. From these results it can be seen that Doustra algorithm takes the longest time to perform the qubit mapping. It also produces the highest amount of SWAP operators by a big margin. Only other qubit mapping algorithm that have used SWAP operators is QiskitAI approach. But the used amount of SWAP operators is approximate 4 times lower than the one used by Doustra. This amount of used SWAP operators also mean that these algorithms uses the biggest amount of CNOT operators. The lowest amount of CNOT operators was produced by the both LightSABRE algorithm versions. For the lowest circuit depth the LightSABRE with decay heuristic produces the best results for the majority of topologies. Interesting result is that for largest topology of Fujitsu Riken, QiskitAI transpiler produced lower depth by 7 % when for Google Willow AI transpiler had two times more larger depth.

7.7. QAOA ansatz

Full table results for the QAOA ansatz are presented in Table 32. The subset of topologies results are presented in the the Table 13. From the results it can be seen that results are similar to

Table 13. Results for QAOA ansatz algorithm

Topology	Optimizer	SWAP	CNOT	Depth	Time (ms)
Google Willow Q105	doustra	22	94	64	95,65
	lightSABRE_decay	1	13	36	22,70
	lightSABRE_lookahead	1	13	36	42,68
	pauliforest	-1	-1	-1	0,00
	qiskit_ai	3	21	63	98,28
	rustiq	1	11	43	26,15
Ibm Eagle Q127	doustra	34	123	73	176,58
	lightSABRE_decay	2	16	41	16,91
	lightSABRE_lookahead	2	16	42	46,30
	pauliforest	-1	-1	-1	0,00
	qiskit_ai	4	24	31	94,57
	rustiq	2	16	39	16,44
Rigetti Novera Q9	doustra	19	84	68	55,43
	lightSABRE_decay	1	11	43	18,72
	lightSABRE_lookahead	1	11	43	36,22
	pauliforest	-1	-1	-1	0,00
	qiskit_ai	2	18	50	85,32
	rustiq	1	13	37	14,33
Riken Fujitsu Q256	doustra	22	87	60	249,53
	lightSABRE_decay	1	13	45	44,52
	lightSABRE_lookahead	1	13	44	72,73
	pauliforest	-1	-1	-1	0,00
	qiskit_ai	4	24	31	126,30
	rustiq	1	11	43	47,48

the QAOA Max Cut Hamiltonian. This can be explained by the fact that to create the QAOA ansatz, the Hamiltonian has to be passed. For the longest execution time, the Doustra algorithm is the slowest one. It also used the highest amount of SWAP and CNOT operators. For the lowest amount of CNOT operators and circuit depth the Rustiq and LightSABRE algorithm produce the best results. For the lowest circuit depth, the Qiskit AI transpiler has performed the best on the majority of circuits (~14 % less).

7.8. Validation on a real quantum computer

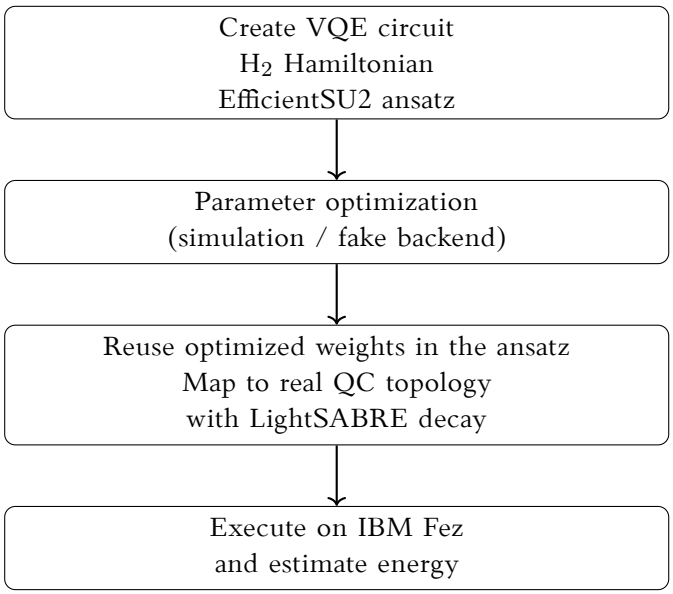

Figure 12. Workflow of the validation on a real quantum computer with a VQE circuit.

The proposed workflow was validated on the IBM Fez backend using the LightSABRE decay mapper and the EfficientSU2 ansatz with one repetition (Figure 12). The test case used the H_2 Hamiltonian with four logical qubits and 16 variational parameters. The optimized parameter vector was transferred from the training stage and bound to the circuit before execution. Selected weights are presented in Table 33. The final run on the real device produced an energy value of 0,9282. After mapping and transpilation, the circuit retained a depth of 16 and a size of 42 gates, confirming that the topology aware mapping preserved a compact circuit structure on hardware. Table 15 summarizes the configuration and the main execution results.

7.8.1. Simulation to hardware energy comparison

To contextualize the real-device result, the hardware energy was compared with the simulation and fake backend baseline obtained under the same methodological pipeline (H_2 Hamiltonian, EfficientSU2 ansatz, and LightSABRE decay mapping). This comparison is necessary to quantify the performance gap introduced by real-device noise and execution conditions.

Table 14. Comparison between simulation/fake-backend and real-hardware VQE energy values.

Environment	**Energy**	**Difference from simulation**
Simulation / Fake backend	0,8199	0,0000
Real hardware (IBM Fez)	0,9282	0,1083

The absolute energy difference is computed as

$$\Delta E = E_{\text{real}} - E_{\text{sim}} = 0{,}9282 - 0{,}8199 = 0{,}1083.$$

The observed increase in energy on real hardware is consistent with expected NISQ-era effects, including gate errors, readout errors, and calibration drift. Therefore, the comparison supports the conclusion that the proposed optimization-and-transfer workflow is functionally valid, while also highlighting the practical degradation when moving from simulation to real quantum hardware.

Table 15. Real-hardware validation results on IBM Fez.

Item	**Value**
Backend	IBM Fez
Backend qubits	156
Mapper	LightSABRE
Ansatz	EfficientSU2
Repetitions	1
Hamiltonian	H_2
Logical qubits	4
Parameters	16
Parameter binding	Verified
Mapped depth	16
Mapped size	42
Energy	0,9282
Estimator precision	0,2
Execution timeout	60 s

8. Qubit mapping algorithm analysis and generalisation propositions

To have a proper analysis of the algorithms and to have a background to write generalisation propositions, the topologies, quantum circuits and qubit mapping features and groups have to be extracted.

8.1. Topologies features

Table 16. Features of the used quantum computer topologies

Topology	Qubits	Average degree	Diameter	Edge density
rigetti_novera_q9	9	2,667	4	0,333
ionq_harmony_q9	9	8,000	1	1,000
ibm_reuschlikon_q16	16	2,750	8	0,183
ibm_tokyo_q20	20	3,500	5	0,184
ibm_poughkeepsie_q20	20	2,300	7	0,121
ibm_almaden_q20	20	2,300	7	0,121
ibm_montreal_q27	27	2,074	12	0,080
ibm_falcon_q27	27	2,074	12	0,080
ibm_cambridge_q28	28	2,143	10	0,079
ibm_rochester_q53	53	2,189	19	0,042
hexagonal_lattice_q54	54	2,667	11	0,050
ibm_manhattan_q65	65	2,215	18	0,035
google_willow_q105	105	3,467	14	0,033
ibm_eagle_q127	127	2,268	26	0,018
ibm_heron_q133	133	2,256	27	0,017
riken_fujitsu_q256	256	3,750	30	0,015

For the analysis of the qubit mapping algorithms performance and possible generalisation based on the quantum computer topologies features, the Table 16 is presented. The features that are calculated for each topology are the following:

- Qubits - number of qubits in the topology.
- Average degree - average number of connections per qubit.
- Diameter - the longest shortest path between any two-qubits in the topology.
- Edge density - ratio of the number of edges to the number of possible edges in the topology.

Table 17. Grouped qubit mapping algorithms based on their features

Algorithm	Group
LightSABRE with lookahead heuristic	Heuristic routing (lookahead)
LightSABRE with decay heuristic	Heuristic routing (decay)
RustiQ	Synthesis-first (low depth)
Doustra	Greedy routing (time-aware)
PauliForest	Synthesis-first (Pauli strings specific)
Qiskit AI-powered transpiler	Machine learning-based

8.2. Qubit mapping algorithms groups

Based on their algorithm descriptions, the group qubit mapping algorithms can be found in Table 17.

8.3. Topology groups

Based on the features presented in Table 16 and the graph structure, the quantum computer topologies can be grouped into the following categories:

- Fully connected topologies (edge density = 1). The only topology in this group is IonQ Harmony.
- Heavy hex (average degree ≈ 2-2,3, edge density ≈ 0,017-0,184). All IBM topologies belong to this group.
- Square, 2D lattice (average degree ≈ 2,6 - 4, edge density ≈ 0,015-0,05). Google Willow, Riken Fujitsu, Hexagonal lattice and Rigetti Novera belong to this group.

The table of each topology with its group is presented in Table 18.

Table 18. Grouped topologies based on their features

Topology	Group
rigetti_novera_q9	Square, 2D lattice
ionq_harmony_q9	Fully connected
ibm_reuschlikon_q16	Heavy hex
ibm_tokyo_q20	Heavy hex
ibm_poughkeepsie_q20	Heavy hex
ibm_almaden_q20	Heavy hex
ibm_montreal_q27	Heavy hex
ibm_falcon_q27	Heavy hex
ibm_cambridge_q28	Heavy hex
ibm_rochester_q53	Heavy hex
hexagonal_lattice_q54	Square, 2D lattice
ibm_manhattan_q65	Heavy hex
google_willow_q105	Square, 2D lattice
ibm_eagle_q127	Heavy hex
ibm_heron_q133	Heavy hex
riken_fujitsu_q256	Square, 2D lattice

8.4. Tested circuits features

Table 19. Features of the tested quantum circuits

Circuit	Qubits	Two-qubit gates	Depth	Diameter	Average degree	Edge density	Interaction variance	Max interactions per edge
H_2	4	30	42	1	3,000	1,000	16,000	9
QAOA	5	6	5	2	2,400	0,600	0,000	1
QAOA ansatz	5	6	7	2	2,400	0,600	0,000	1
EfficientSU2	5	4	8	4	1,600	0,400	0,000	1
random	9	60	73	4	2,889	0,361	6,391	8
LiH	12	6023	7328	2	9,667	0,879	23859,821	474

For the analysis of the qubit mapping algorithms performance and possible generalisation based on the quantum circuits features, the Table 19 is presented. The features that are calculated for each circuit are the following:

- Qubits - number of qubits in the circuit.
- Two-qubit gates - number of two-qubit gates in the circuit.
- Depth - depth of the circuit.
- Diameter - the longest shortest path between any two-qubits in the circuit interaction graph.
- Average degree - average number of connections per qubit in the circuit interaction graph.
- Edge density - ratio of the number of edges to the number of possible edges in the circuit interaction graph.
- Interaction variance - variance of the number of interactions per edge in the circuit interaction graph.
- Max interactions per edge - maximum number of interactions per edge in the circuit interaction graph.

8.5. Selected criteria

For the generalisation propositions, the following criteria of the circuit optimisation are selected:

- CX gates count - number of CNOT gates in the circuit after the qubit mapping. The goal is to reduce the number of CNOT gates as much as possible. This allows to minimise the error rate of the circuit execution on the quantum computer.
- Circuit depth - depth of the circuit after the qubit mapping. The goal is to reduce the circuit depth as much as possible, because the shorter time it takes to execute the circuit, the less probability of decoherence errors and allows to maintain the quantum state fidelty.

Each criteria will be evaluated **separately** because they are not tightly related to each other and it would make generalisation much harder with less practical results. Also, it was decided to not included the number of qubits as a success criteria because the number of logical qubits is defined by the quantum algorithm and cannot be changed by the qubit mapping algorithm.

8.6. Topology and circuit pattern analysis

Based on the analysis of the results presented in the previous section, the following patterns will be analysed based on the topology groups and circuit features.

8.6.1. Optimisation for CNOT gates count

Fully connected topologies. Key findings for the Pauli strings input:

- For dense Pauli strings (like H_2 Hamiltonian or demo random strings), the Pauli string specific synthesis algorithms (like PauliForest) are the most suited for this topology:

  - H_2 Hamiltonian and demo: PauliForest is the best algorithm. Since the topology is fully connected, the advantage comes not from routing, but from the algorithm's ability to synthesize gates directly from the operator list, minimizing the base gate count compared to standard synthesis methods.
- For sparse Hamiltonians (like QAOA Hamiltonian), the choice of algorithm is statistically insignificant:
  - QAOA Hamiltonian: results show a tie between synthesis (Rustiq), routing (lookahead-/decay), and AI. On a fully connected graph, the routing overhead is zero, meaning standard decomposition methods perform identically regardless of the mapping strategy.

Key findings for the quantum circuits input:

- For variational ansatz circuits, the specific mapping algorithm has little impact on the CNOT count:
  - EfficientSU2 and QAOA ansatz: all major algorithm groups (Rustiq, lookahead, Decay, AI) produce comparable results. With full connectivity, the optimization problem reduces to standard gate synthesis, eliminating the routing constraints that usually differentiate these algorithms.

Heavy hex topologies. Key findings for the Pauli strings input:

- For the small, dense Hamiltonians (like H_2 Hamiltonian), the circuit synthesis and mapping algorithms (like PauliForest) are the most suited for this topology group:
  - H_2 Hamiltonian: PauliForest is the best algorithm for the majority of heavy hex topologies (IBM Almaden, Cambridge, Eagle, Heron, Poughkeepsie, Tokyo) (see Table 27).
- For the sparse Hamiltonians (like QAOA Hamiltonian) or deep Hamiltonians (like LiH), the heuristic routing algorithms (lookahead or decay) are often the most suited, though synthesis-first approaches are competitive on newer devices:
  - QAOA Hamiltonian: decay or lookahead heuristics provide the best results on IBM Almaden, Cambridge, and Poughkeepsie (see Table 30).
  - LiH Hamiltonian: PauliForest is best on IBM Almaden and Tokyo, while LightSABRE lookahead is best on IBM Cambridge, Eagle, Falcon, Heron, Manhattan, Montreal, Poughkeepsie, Reuschlikon, and Rochester (see Table 29).

Key findings for the quantum circuits input:

- The synthesis-first algorithms (like Rustiq) are the most suited for variational ansatz circuits on this topology group:
  - QAOA ansatz: Rustiq produces the lowest CNOT count for IBM Eagle, Falcon, and Rochester (see Table 32).
  - EfficientSU2 ansatz: the results are mixed between synthesis (Rustiq) and lookahead heuristics, with Rustiq performing best on IBM Eagle and Poughkeepsie (tied with decay) (see Table 31).

Square, 2D lattice topologies. Key findings for the Pauli strings input:

- For the Hamiltonians with low interaction variance (like QAOA Hamiltonian), the synthesis-first algorithms (like Rustiq) are the most suited for this topology group, especially as the lattice size increases:
  - QAOA Hamiltonian: Rustiq is the best algorithm for both the medium-scale Google Willow (105 qubits) and large-scale Riken Fujitsu (256 qubits) topologies (see Table 30).
- For small dense Hamiltonians (like H_2), the topology size influences the generalization:
  - H_2 Hamiltonian: decay heuristic is best for small lattices (Rigetti Novera), while Synthesis (Rustiq) becomes best for the largest lattice (Riken Fujitsu) (see Table 27).
- For deep Hamiltonian simulations (like LiH Hamiltonian), the lattice scale determines the optimal approach:
  - LiH Hamiltonian: PauliForest is the most suited for Google Willow, Hexagonal Lattice, and Riken Fujitsu, minimizing CNOT count more effectively than routing heuristics on these square and lattice topologies (see Table 29).

Key findings for the quantum circuits input:

- For the QAOA ansatz, the synthesis-first algorithms (like Rustiq) are the most suited for this topology group:
  - QAOA ansatz: Rustiq consistently produces the lowest CNOT count for Google Willow and Riken Fujitsu (see Table 32).
- For the EfficientSU2 ansatz, the lookahead heuristic based algorithms are the most suited for this topology group:
  - EfficientSU2 ansatz: lookahead routing is the best algorithm for Rigetti Novera and Riken Fujitsu, outperforming synthesis methods which struggle to optimize the specific hardware-efficient structure of the SU2 ansatz on square grids (see Table 31).

8.6.2. Optimisation for reducing circuit depth

Fully connected topologies. Key findings for the Pauli strings input:

- The Pauli string specific synthesis algorithms (like PauliForest) are the most suited for this topology for all tested Hamiltonian inputs:
  - H_2, QAOA Hamiltonian, and Demo: PauliForest consistently yields the lowest depth. This indicates that for Hamiltonian simulation on fully connected hardware, synthesizing the circuit to maximize parallel execution of commuting Pauli terms is more effective than mapping a pre-compiled circuit (see Table 27, 30, 28).

Key findings for the quantum circuits input:

- For QAOA ansatzes, synthesis-first algorithms (like Rustiq) are the most suited for this topology:
  - Rustiq produces the best depth. Since no SWAP gates are required, Rustiq ability to

resynthesize the circuit provides a depth advantage over heuristic routing methods which place the existing gates (see Table 32).

- For hardware-efficient ansatzes (like EfficientSU2), the choice of algorithm is less differentiating:
  - EfficientSU2: the results are split between synthesis (Rustiq) and heuristic routing (lookahead/decay), suggesting that optimisation algorithm logic has less impact on this topology group (see Table 31).

Heavy hex topologies. Key findings for the Pauli strings input:

- For the majority of Pauli string inputs, the Pauli string specific synthesis algorithms (like PauliForest) are the most suited for this topology group:
  - H_2 Hamiltonian and Demo random Pauli strings: PauliForest is the best algorithm for every single heavy hex topology (from IBM Almaden to IBM Tokyo). The algorithm synthesis helps to reduce the circuit depth (see Table 27, 28).
  - LiH Hamiltonian: PauliForest is the best performing algorithm across all heavy hex topologies (IBM Almaden, Cambridge, Eagle, Falcon, Heron, Manhattan, Montreal, Poughkeepsie, Reuschlikon, Rochester, and Tokyo), offering significant depth reduction compared to routing algorithms (see Table 29).
- For the QAOA Hamiltonian, the machine learning-based algorithms (like Qiskit AI-powered transpiler) are the most suited for this topology group:
  - QAOA Hamiltonian: The AI transpiler is the best algorithm for the majority of chips (IBM Almaden, Cambridge, Eagle, Heron, Manhattan, Poughkeepsie, Rochester), suggesting that machine learning models are capable of reducing circuit depth on heavy hex structures (see Table 30).

Key findings for the quantum circuits input:

- For the QAOA ansatz, the Machine learning-based algorithms are the most suited for this topology group:
  - QAOA ansatz: the AI transpiler produces the lowest circuit depth on almost all heavy hex architectures, outperforming both heuristic routing and synthesis-first methods (see Table 32).
- For the EfficientSU2 ansatz, the results are mixed, with machine learning algorithms being best on large devices (IBM Rochester) and heuristic/synthesis methods (lookahead, decay, Rustiq) performing better on smaller devices (IBM Falcon, IBM Montreal) (see Table 31).

Square, 2D lattice topologies. Key findings for the Pauli strings input:

- For dense or random Pauli strings, the Pauli string specific synthesis algorithms are the most suited for this topology group:
  - H_2 Hamiltonian and demo random Pauli strings: PauliForest is the best algorithm across all square lattice sizes (Rigetti Novera, Google Willow, Riken Fujitsu) (see Table 27, 28).

- For sparse Hamiltonians (QAOA), the topology scale determines the best approach:
  - QAOA Hamiltonian: PauliForest is best for smaller lattices (Rigetti, Google Willow), but the machine learning algorithms are most suited for the largest lattice (Riken Fujitsu) (see Table 30).
- For deep Hamiltonians (LiH), the results vary significantly by device:
  - LiH Hamiltonian: PauliForest is best for Google Willow, Hexagonal Lattice, and Riken Fujitsu (see Table 29).

Key findings for the quantum circuits input:

- For the largest square lattice topologies, the Machine learning-based algorithms are the most suited:
  - QAOA ansatz and EfficientSU2 ansatz: On the Riken Fujitsu (256 qubits) topology, the AI transpiler achieves the best circuit depth (see Table 32, 31).
- For medium and small square lattices, heuristic and synthesis methods prevail:
  - QAOA ansatz: decay/lookahead is best for Google Willow, while Rustiq is best for Rigetti Novera.
  - EfficientSU2 ansatz: lookahead is best for Google Willow, while decay/Rustiq is best for Rigetti Novera (see Table 32, 31).

8.7. Decision trees based on the generalisation propositions

Based on the key findings that were proposed in the Section 8.6, the generalisation rules can be created by using decision trees.

8.7.1. Optimising the CNOT count

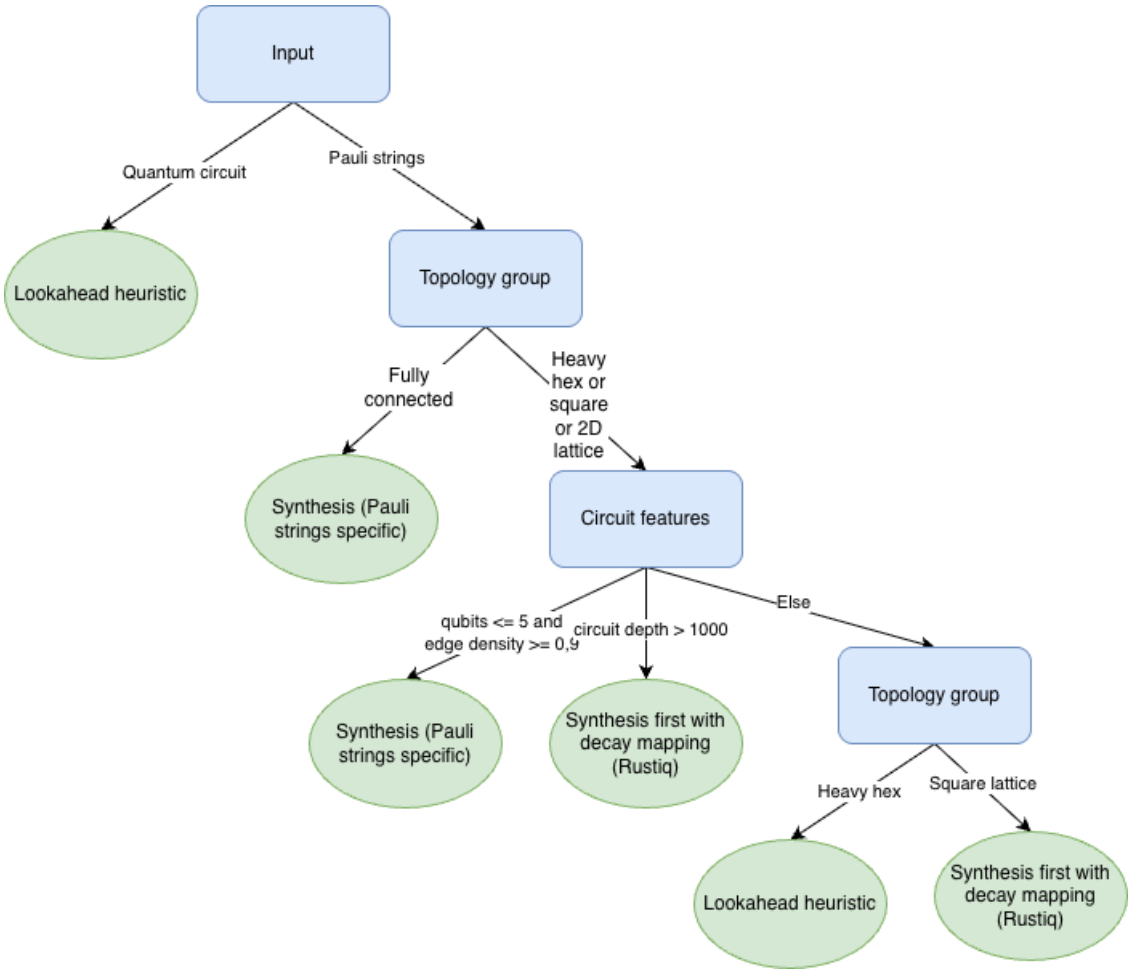

Figure 13. Decision tree for selecting the qubit mapping algorithm based on CNOT count optimisation

Based on the generalisation propositions, the decision tree for selecting the qubit mapping algorithm based on CNOT count optimisation is presented in the Figure 13. It can be seen that the primary decision node is the type of input. For the quantum circuit, the solution is to use a lookahead heuristic algorithm (like LightSABRE variation). For the Pauli strings, the important factor is if the topology is fully connected or not. If it is, then the synthesis algorithms with Pauli strings specifics are the most effective ones. If it is constrained layout, for the smaller but dense quantum circuits the synthesis algorithms with Pauli strings specifics are also the most suited. For the large quantum circuits the synthesis first with decay mapping works the best. For the other cases, lookahead heuristic algorithms are used for heavy hex group topologies and synthesis first with decay mapping is for others.

8.7.2. Optimising the circuit depth

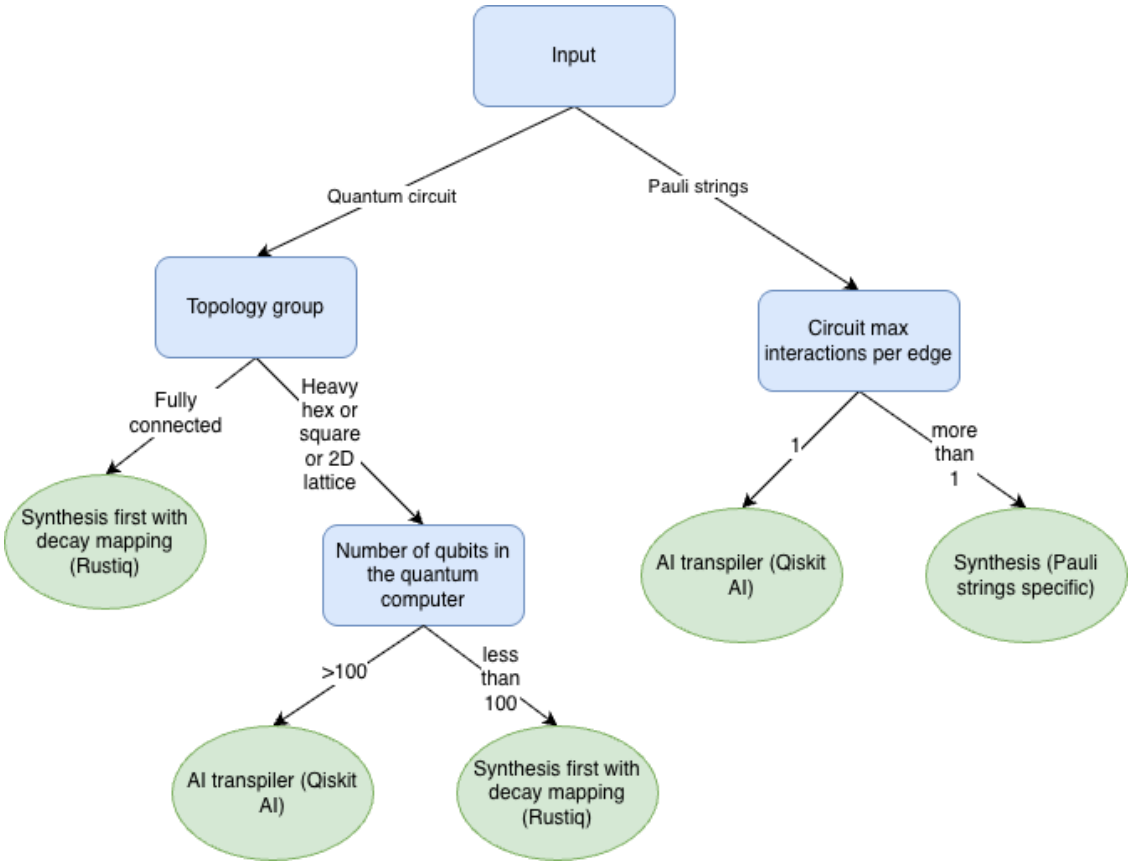

Figure 14. Decision tree for selecting the qubit mapping algorithm based on circuit depth optimisation

Based on the generalisation propositions, the decision tree for selecting the qubit mapping algorithm based on circuit depth optimisation is presented in Figure 14. If the input is the quantum circuit, then the topology group is analysed. If it is a fully connected graph, then synthesis first with decay mapping (like Rustiq) should be used. For other structures, if it is a larger topology, the AI transpiler performs the best, for other cases the synthesis first with decay mapping (like Rustiq). For the Pauli strings input, the quantum circuit max interactions per graph edge ar checked. If the same qubit pairs interact only once, then the AI transpiler works the best. For other cases synthesis algorithms with Pauli strings specifics are the most effective ones.

8.8. Generalisation rules testing

The generalisation rules for CNOT and circuit depth criteria were tested. For each topology and circuit combination the best algorithm was determined by running brute force check of all algorithms. If the proposed algorithm of the decision tree is the same as the one as the identified as the best one, it is marked as a correct test result. Generalisation rules accuracy can be seen in a Table 20. Full results tables for CNOT and depth evaluations are presented in Table 34 and 35 respectively. Accuracy of generalisation rules is better than random baseline of 16,7 %. However, it can be seen that for CNOT count success criteria the rules need refinement. Results can be

explained by the fact, that with the current set of experiment circuits it is hard to see the tendencies of the algorithms in their performance. Also defined rules are needed for the circuit input to have better results. Generalisation rules for circuit depth have a higher accuracy. This can be explained by the fact that the same qubit mapping algorithm performs the best for almost all topologies for the specific quantum circuit. Also, it is hard to test the performance of the algorithms because majority of them work randomly and their result depends on the seed it got.

Success criteria	Correct selections	Incorrect selections	Accuracy
CNOT count	43	58	45,74 %
Circuit depth	68	26	72,34 %

Table 20. Accuracy of the generalisation rules

8.9. Hypothesis validation and limitations of the current generalisation rules

The decision-tree rules indicate that the hypothesis is not rejected: generalisation of scheme optimisation across different topologies appears feasible under the CNOT count or circuit depth criteria (see Figures 13, 14). The CNOT decision tree achieves 45,74 % accuracy (approximately 2,7 times above the random baseline of 16,67 %), and the circuit depth tree achieves 72,34 % accuracy (approximately 4,3 times above random). Both results demonstrate that topology class and circuit structure are meaningful predictors of mapper performance. However, the current rule quality is not yet sufficient for robust practical guidance.

This has a practical implication in the current NISQ era. Quantum hardware execution is expensive - both financially and in terms of limited access to them. Circuit optimisation time on the classical computer is negligible. Thus, running a brute force method iterating through all qubit mapping algorithms to find the one with the lowest value for the selected criterion is a rational decision for any real quantum workload. Penalty for choosing suboptimal mapper costs a lot more than saving computation time of classical compute.

The following limitations of the current rules are noted:

- The rules are based on a limited set of quantum computer topologies and quantum circuits, with relatively few deep circuits and high qubit count instances. They therefore capture the behaviour of the selected benchmark set, but they do not yet justify universal recommendations.
- The generalisation rules are based on the specific implementations of the qubit mapping algorithms. Customisations of the qubit mapping algorithms for the specific topologies or circuits may lead to different results.

Therefore, practical mapper-selection recommendations cannot yet be provided with sufficient confidence, and the rules should not be treated as production ready. These trees confirm that topology class and circuit structure are meaningful predicators of mapper performance. This

motivates future work on more robust rules analysis with broader benchmarks. For practical use it is recommended to use brute force approach.

8.10. Investigation under additional error correction complexity

Given the scope constraints of this thesis and the primary focus on qubit mapping algorithm comparison, this objective is addressed through a targeted scoping experiment rather than a full benchmark. The goal is to establish whether mitigation techniques introduce meaningful overhead in the mapping pipeline and to validate that the framework supports mitigation workflows, rather than to provide definitive hardware-level performance claims. A full benchmark across all circuits, topologies, and mappers is identified as future work

The test scenario is:

- Circuits: one Hamiltonian-based circuit (H_2) and one ansatz-based circuit (EfficientSU2).
- Topologies: one fully connected topology (IonQ Harmony) and one constrained topology (Rigetti Novera).
- Mappers: LightSABRE decay and lookahead heuristics versions.
- Mitigation profile: comparison between `Off` and `On (attempted)`. The enabled profile attempts dynamic decoupling at circuit level and enables Pauli twirling and T-REx estimator settings in the local simulation workflow.

The goal of this experiment is to quantify complexity overhead and sensitivity, not to provide a full benchmark. Therefore, the analysis focuses on:

- optimisation runtime overhead;
- circuit complexity overhead (CNOT count and circuit depth);
- one quality indicator: relative expectation-value error (`ErrRate`).

Table 21. Ansatz (EfficientSU2) mitigation complexity results averaged over 5 runs

Mapper	Topology	Mitigation	Time (ms)	CNOT	Depth	ErrRate
lightSABRE_lookahead	ionq_harmony_q9	Off	314,054	3	15	0,020
lightSABRE_lookahead	ionq_harmony_q9	On	349,691	3	15	0,021
lightSABRE_lookahead	rigetti_novera_q9	Off	64,560	3	15	0,021
lightSABRE_lookahead	rigetti_novera_q9	On	109,495	3	15	0,020
lightSABRE_decay	ionq_harmony_q9	Off	340,103	3	11	0,023
lightSABRE_decay	ionq_harmony_q9	On	352,932	3	11	0,021
lightSABRE_decay	rigetti_novera_q9	Off	65,579	3	23	0,027
lightSABRE_decay	rigetti_novera_q9	On	66,839	3	23	0,022

Table 22. Hamiltonian (H_2) mitigation complexity results averaged over 5 runs

Mapper	Topology	Mitigation	Time (ms)	CNOT	Depth	ErrRate
lightSABRE_lookahead	ionq_harmony_q9	Off	358,442	36	70	0,023
lightSABRE_lookahead	ionq_harmony_q9	On	407,193	36	70	0,017
lightSABRE_lookahead	rigetti_novera_q9	Off	137,756	45	79	0,020
lightSABRE_lookahead	rigetti_novera_q9	On	124,441	45	79	0,021
lightSABRE_decay	ionq_harmony_q9	Off	371,195	36	55	0,021
lightSABRE_decay	ionq_harmony_q9	On	338,209	36	55	0,020
lightSABRE_decay	rigetti_novera_q9	Off	133,417	45	148,6	0,021
lightSABRE_decay	rigetti_novera_q9	On	87,273	44,2	150,2	0,064

This experiment results shows that mitigation impact depends on configuration rather than uniform. In several cases, error decreases under the enabled profile, while in others it remains similar or increases. Circuit structure metrics are generally stable across mitigation modes, whereas runtime differences are non-monotonic in local simulation and should be interpreted cautiously. Therefore, these results are used as evidence for method validation within this thesis scope, rather than as final hardware-level performance claims.

9. Real-world use case

9.1. Angle PQK SVM use case

The Angle PQK SVM method and the associated tabular data baseline were originally developed in the CLARYON [Pap26; PVS+26] project, by researchers at the Medical University of Vienna. In this thesis, we collaborated with these scientists, and the method was adopted and evaluated within the QuMeld framework. The contribution of this work is the hardware-aware integration, execution, and comparative testing pipeline across multiple backend topologies and mapping strategies. The obtained results were shared with the CLARYON author as part of the collaboration.

9.1.1. Angle-Encoded Projected Quantum Kernel SVM

The Angle-Encoded Projected Quantum Kernel SVM (Angle-PQK SVM) is a hybrid quantum-classical classifier that uses a quantum circuit to transform input features into a richer representation, over which a classical SVM is then trained.

Angle encoding maps each feature x_i of an input $x \in \mathbb{R}^n$ onto a dedicated qubit by rotating it around the Y-axis of the Bloch sphere, whose poles represent the classical states $|0\rangle$ and $|1\rangle$ and every other point a quantum superposition:

$$\mathrm{RY}(c \cdot x_i)\,|0\rangle, \tag{5}$$

where $c > 0$ is a bandwidth hyperparameter controlling encoding sensitivity. This requires exactly n qubits with no entanglement, and unlike amplitude encoding, preserves per-feature magnitude information without any normalisation [TWC+24].

The position of each qubit on the Bloch sphere is then described by three expectation values along its orthogonal axes, known as Pauli operators. For qubit i:

$$v_i(x) = \big(\langle X_i\rangle, \langle Y_i\rangle, \langle Z_i\rangle\big). \tag{6}$$

Concatenating across all qubits yields a classical Pauli feature vector [HBM+21]:

$$\phi(x) = \big(v_1(x), \ldots, v_n(x)\big) \in \mathbb{R}^{3n}, \tag{7}$$

which captures non-linear relationships through the trigonometric nature of the RY rotations.

An RBF kernel is computed over the Pauli feature vectors:

$$K(x,x') = \exp\Big(-\gamma \left\|\phi(x) - \phi(x')\right\|^2\Big), \tag{8}$$

where $\gamma = \big(3n \cdot \mathrm{Var}(\phi)\big)^{-1}$ by default. The bandwidth c is the most impactful hyperparameter, directly controlling the discriminativeness of the kernel [SW22]. The resulting kernel matrix is passed to a soft-margin SVM to find the optimal decision boundary.

9.1.2. Running the experiment

Experiments were executed as a nested sweep with the following structure:

1. Binary tabular datasets were prepared by numerical encoding, followed by Z-score normalization with parameters fitted on the training partition and applied unchanged to the corresponding test partition.
2. Two Angle PQK circuit variations were evaluated for each dataset:
   - non-entangling variant (baseline),
   - entangling variant with CX-based layers.
3. For each dataset-variant pair, stratified multi-seed k-fold cross-validation was performed to obtain robust estimates.
4. For each fold, representative circuits were compiled and routed with QuMeld for each target backend topology. Mapper selection was performed under coupling-map constraints using standardized circuit-quality criteria.
5. Learning metrics (test accuracy, balanced accuracy, gain over majority baseline, support vectors) and mapping metrics (depth, CX count, SWAP count, mapping time) were collected jointly.
6. Results were aggregated across folds and seeds, then compared across backends and across both Angle PQK variations to identify configurations that are both predictive and hardware-feasible.

9.1.3. Iris benchmark results

To illustrate the behaviour of the above workflow on a controlled benchmark, this section reports results on the binary Iris dataset [Fis36]. Results are presented in Tables 23, 24.

Table 23. Angle PQK SVM sweep summary for Iris dataset and no entangling

Backend	Qubits	Mapper	Status	Depth	CX	SWAP	Train acc	Test acc
rigetti_novera_q9	9	rustiq	ok	5	0	0	1,0000	1,0000
ibm_reuschlikon_q16	16	rustiq	ok	5	0	0	1,0000	1,0000
ibm_almaden_q20	20	lightSABRE_decay	ok	5	0	0	0,9994	1,0000
ibm_paughkeepsie_q20	20	rustiq	ok	5	0	0	1,0000	1,0000
ibm_falcon_q27	27	rustiq	ok	5	0	0	1,0000	1,0000
ibm_montreal_q27	27	rustiq	ok	5	0	0	1,0000	1,0000
ibm_cambridge_q28	28	rustiq	ok	5	0	0	0,9989	1,0000
ibm_tokyo_q20	20	rustiq	ok	5	0	0	0,9994	1,0000
ibm_rochester_q53	53	rustiq	mapping-only	5	0	0	NA	NA
hexagonal_lattice_q54	54	rustiq	mapping-only	5	0	0	NA	NA
ibm_manhattan_q65	65	rustiq	mapping-only	5	0	0	NA	NA
ionq_harmony_q9	9	rustiq	ok	5	0	0	0,9994	1,0000
ibm_eagle_q127	127	rustiq	mapping-only	5	0	0	NA	NA
ibm_heron_q133	133	rustiq	mapping-only	5	0	0	NA	NA
google_willow_q105	105	rustiq	mapping-only	5	0	0	NA	NA
riken_fujitsu_q256	256	rustiq	mapping-only	5	0	0	NA	NA

Table 24. Angle PQK SVM sweep summary for Iris dataset and entangling

Backend	Qubits	Mapper	Status	Depth	CX	SWAP	Train acc	Test acc
rigetti_novera_q9	9	lightSABRE_decay	ok	12	7	0	0,8500	0,8533
ibm_reuschlikon_q16	16	lightSABRE_decay	ok	12	7	0	0,8533	0,8533
ibm_almaden_q20	20	lightSABRE_decay	ok	12	7	0	0,8494	0,8333
ibm_paughkeepsie_q20	20	lightSABRE_decay	ok	12	7	0	0,8550	0,8289
ibm_falcon_q27	27	rustiq	ok	11	7	0	0,8517	0,8533
ibm_montreal_q27	27	lightSABRE_decay	ok	12	7	0	0,8572	0,8467
ibm_cambridge_q28	28	rustiq	ok	11	7	0	0,8533	0,8378
ibm_tokyo_q20	20	lightSABRE_decay	ok	12	7	0	0,8539	0,8356
ibm_rochester_q53	53	lightSABRE_decay	mapping-only	12	7	0	NA	NA
hexagonal_lattice_q54	54	lightSABRE_decay	mapping-only	12	7	0	NA	NA
ibm_manhattan_q65	65	lightSABRE_decay	mapping-only	12	7	0	NA	NA
ionq_harmony_q9	9	rustiq	ok	11	7	0	0,8578	0,8400
ibm_eagle_q127	127	lightSABRE_decay	mapping-only	12	7	0	NA	NA
ibm_heron_q133	133	lightSABRE_decay	mapping-only	12	7	0	NA	NA
google_willow_q105	105	lightSABRE_decay	mapping-only	12	7	0	NA	NA
riken_fujitsu_q256	256	lightSABRE_decay	mapping-only	12	7	0	NA	NA

In the Iris benchmark, the no-entangling variant reached up to 1,0000 train accuracy and 1,0000 test accuracy, while the entangling variant peaked at 0,8578 train accuracy and 0,8599 test accuracy. The no-entangling variant therefore performed better across the tested backends.

9.1.4. Wisconsin Breast Cancer benchmark results

To extend the evaluation to a second binary classification benchmark, this section reports results on the Wisconsin Breast Cancer dataset [WSM94]. Results are presented in Tables 25, 26.

Table 25. Angle PQK SVM sweep summary for Wisconsin Breast Cancer dataset with no entanglement

Backend	Qubits	Mapper	Status	Depth	CX	SWAP	Train acc	Test acc
rigetti_novera_q9	9	rustiq	ok	5	0	0	0,9684	0,9596
ibm_reuschlikon_q16	16	rustiq	ok	5	0	0	0,9704	0,9614
ibm_almaden_q20	20	rustiq	ok	5	0	0	0,9698	0,9502
ibm_paughkeepsie_q20	20	rustiq	ok	5	0	0	0,9697	0,9637
ibm_falcon_q27	27	rustiq	ok	5	0	0	0,9698	0,9596
ibm_montreal_q27	27	rustiq	ok	5	0	0	0,9720	0,9590
ibm_cambridge_q28	28	rustiq	ok	5	0	0	0,9703	0,9614
ibm_tokyo_q20	20	rustiq	ok	5	0	0	0,9690	0,9608
ibm_rochester_q53	53	rustiq	mapping-only	5	0	0	NA	NA
hexagonal_lattice_q54	54	rustiq	mapping-only	5	0	0	NA	NA
ibm_manhattan_q65	65	rustiq	mapping-only	5	0	0	NA	NA
ionq_harmony_q9	9	rustiq	ok	5	0	0	0,9675	0,9572
ibm_eagle_q127	127	rustiq	mapping-only	5	0	0	NA	NA
ibm_heron_q133	133	rustiq	mapping-only	5	0	0	NA	NA
google_willow_q105	105	rustiq	mapping-only	5	0	0	NA	NA
riken_fujitsu_q256	256	rustiq	mapping-only	5	0	0	NA	NA

Table 26. Angle PQK SVM sweep summary for Wisconsin Breast Cancer dataset with entanglement

Backend	Qubits	Mapper	Status	Depth	CX	SWAP	Train acc	Test acc
rigetti_novera_q9	9	lightSABRE_decay	ok	12	7	0	0,8757	0,8599
ibm_reuschlikon_q16	16	lightSABRE_decay	ok	12	7	0	0,8770	0,8494
ibm_almaden_q20	20	lightSABRE_decay	ok	12	7	0	0,8760	0,8511
ibm_paughkeepsie_q20	20	lightSABRE_decay	ok	12	7	0	0,8726	0,8529
ibm_falcon_q27	27	rustiq	ok	11	7	0	0,8732	0,8587
ibm_montreal_q27	27	rustiq	ok	11	7	0	0,8773	0,8541
ibm_cambridge_q28	28	lightSABRE_decay	ok	12	7	0	0,8698	0,8523
ibm_tokyo_q20	20	lightSABRE_decay	ok	12	7	0	0,8732	0,8500
ibm_rochester_q53	53	lightSABRE_decay	mapping-only	12	7	0	NA	NA
hexagonal_lattice_q54	54	lightSABRE_decay	mapping-only	12	7	0	NA	NA
ibm_manhattan_q65	65	lightSABRE_decay	mapping-only	12	7	0	NA	NA
ionq_harmony_q9	9	rustiq	ok	11	7	0	0,8686	0,8529
ibm_eagle_q127	127	lightSABRE_decay	mapping-only	12	7	0	NA	NA
ibm_heron_q133	133	lightSABRE_decay	mapping-only	12	7	0	NA	NA
google_willow_q105	105	rustiq	mapping-only	11	7	0	NA	NA
riken_fujitsu_q256	256	lightSABRE_decay	mapping-only	12	7	0	NA	NA

In the Wisconsin Breast Cancer benchmark, the non-entangling variant reached up to 0,9720 train accuracy and 0,9637 test accuracy, while the entangling variant peaked at 0,8773 train accuracy and 0,8599 test accuracy. The no-entangling variant therefore remained stronger across the tested backends.

Both benchmarks confirm that the QuMeld framework supports end-to-end quantum machine learning workflows under realistic hardware constraints. The no-entangling variant

consistently outperformed the entangling variant across both datasets and all tested backend topologies. This shows that for these classification tasks the additional circuit complexity introduced by entangling layers does not improve predictive performance and increases mapping overhead unnecessarily. The successful execution across multiple topology types and mapping strategies demonstrates that the framework is practically usable for quantum machine learning problems beyond synthetic benchmarks.

10. Results and conclusions

Results:

1. Proposed and open sourced a modular qubit mapping testing framework for topology-aware circuit optimisation (Section 5).
2. Ran experiments for the selected VQE and QAOA benchmark circuits and obtained the following results:
   (a) VQE H_2 Hamiltonian: within the fully connected group, the best observed result was from PauliForest on IonQ Harmony, with CNOT count 22, depth 30, and execution time 17,25 ms. Within the heavy hex group, the best observed result was from PauliForest on IBM Eagle, with CNOT count 40, depth 38, and execution time 765,43 ms. Within the square and 2D lattice group, the lowest observed CNOT count was 39 (LightSABRE lookahead and Rustiq tie on Riken Fujitsu), while the lowest observed depth was 38 (PauliForest on Riken Fujitsu in 6058,29 ms) (Table 27).
   (b) VQE random Hamiltonian: within the fully connected group, the best observed result was from PauliForest on IonQ Harmony, with CNOT count 45, depth 38, and execution time 31,91 ms. Within the heavy hex group, the best observed result was from PauliForest on IBM Eagle, with CNOT count 78, depth 46, and execution time 789,47 ms. Within the square and 2D lattice group, the lowest observed CNOT count was 70 (LightSABRE lookahead on Riken Fujitsu), while the lowest observed depth was 47 (PauliForest on Riken Fujitsu in 6040,20 ms) (Table 28).
   (c) VQE LiH Hamiltonian: within the fully connected group, this benchmark was not executable on IonQ Harmony and results are marked as -1. Within the heavy hex group, the lowest observed CNOT count was 5722 (PauliForest on IBM Tokyo in 1070,47 ms), while the lowest observed depth was 3136 (PauliForest on IBM Tokyo in 1070,47 ms). Within the square and 2D lattice group, the lowest observed CNOT count was 7247 (PauliForest on Google Willow in 1564,57 ms), while the lowest observed depth was 3282 (PauliForest on Riken Fujitsu in 7116,02 ms) (Table 29).
   (d) QAOA Max-Cut Hamiltonian: within the fully connected group, the lowest observed CNOT count and depth were both tied at 12 and 15 on IonQ Harmony for LightSABRE decay, LightSABRE lookahead, Qiskit AI, and Rustiq, with execution times 45,23 ms, 54,16 ms, 83,47 ms, and 43,26 ms respectively. Within the heavy hex group, the lowest observed CNOT count was 16 (LightSABRE decay and LightSABRE lookahead tie on IBM Eagle in 18,00 ms and 43,06 ms), while the lowest observed depth was 17 (Qiskit AI on IBM Eagle in 93,16 ms). Within the square and 2D lattice group, the lowest observed CNOT count was 9 (Rustiq on Riken Fujitsu in 55,51 ms), while the lowest observed depth was 18 (Qiskit AI on Riken Fujitsu in 129,15 ms) (Table 30).
   (e) VQE EfficientSU2 ansatz: within the fully connected group, the lowest observed CNOT count and depth were both tied at 4 and 12 on IonQ Harmony for

LightSABRE decay, LightSABRE lookahead, and Rustiq, with execution times 49,69 ms, 58,24 ms, and 47,96 ms respectively. Within the heavy hex group, the lowest observed CNOT count was 3 (Rustiq on IBM Eagle in 11,49 ms), while the lowest observed depth was 12 (LightSABRE lookahead on IBM Eagle in 31,17 ms). Within the square and 2D lattice group, the lowest observed CNOT count was 2 (LightSABRE lookahead on Riken Fujitsu in 93,78 ms), while the lowest observed depth was 16 (Qiskit AI on Riken Fujitsu in 120,78 ms) (Table 31).

(f) QAOA ansatz: within the fully connected group, the lowest observed CNOT count and depth were both tied at 12 and 23 on IonQ Harmony for LightSABRE decay, LightSABRE lookahead, and Rustiq, with execution times 52,69 ms, 61,30 ms, and 52,24 ms respectively. Within the heavy hex group, the lowest observed CNOT count was 16 (LightSABRE decay, LightSABRE lookahead, and Rustiq tie on IBM Eagle in 16,91 ms, 46,30 ms, and 16,44 ms), while the lowest observed depth was 31 (Qiskit AI on IBM Eagle in 94,57 ms). Within the square and 2D lattice group, the lowest observed CNOT count was 11 (Rustiq on Riken Fujitsu in 47,48 ms), while the lowest observed depth was 31 (Qiskit AI on Riken Fujitsu in 126,30 ms) (Table 32).

3. Preliminary decision tree rules for mapper selection achieved 2,7 – 4,3 times above the random baseline accuracy (16,7 %), confirming that topology class and circuit structure are meaningful predictors of mapper performance (Table 20).
4. Real life use cases of an Angle-PQK SVM: the framework was used to support topology-aware quantum machine learning experiments for two binary classification benchmarks. On the Iris dataset, the no-entangling variant reached 1,0000 train accuracy and 1,0000 test accuracy, while the entangling variant peaked at 0,8578 train accuracy and 0,8599 test accuracy (Table 23). On the Wisconsin Breast Cancer dataset, the no-entangling variant reached up to 0,9720 train accuracy and 0,9637 test accuracy, while the entangling variant peaked at 0,8773 train accuracy and 0,8599 test accuracy (Tables 25, 26).
5. Testing a VQE with EffectiveSU2 and H_2 Hamiltonian on a real quantum computer showed that the framework can be applied beyond simulated backends, and that results are close to simulated ones (Table 15).

Conclusions:

1. Quantum optimisation algorithms do not reduce the qubit mapping problem to a simplified form and must be treated as general circuits during mapping.
2. No single qubit-mapping algorithm is globally optimal, including default framework mappers. Criterion-driven exhaustive evaluation across all candidate qubit mapping algorithms is the recommended practical strategy in the NISQ era.
3. For Hamiltonian simulation workloads where minimising circuit depth is the primary criterion, Pauli-string-aware synthesis should be prioritised over routing-based approaches in mapper selection.
4. The proposed framework demonstrates that qubit mapping strategies can be benchmarked

reproducibly across different quantum computer topologies under unified evaluation criteria.

5. Real-world validation on two benchmark datasets and real IBM hardware demonstrated practical end-to-end usability of the framework beyond synthetic benchmarks.

References

[AAA+24] R. Acharya, D. A. Abanin, L. Aghababaie-Beni, I. Aleiner, et al. Quantum error correction below the surface code threshold. *Nature*. 2024. ISBN 1476-4687. Available from: https://doi.org/10.1038/s41586-024-08449-y.

[ACH23] R. Au-Yeung, N. Chancellor, P. Halffmann. NP-hard but no longer hard to solve? Using quantum computing to tackle optimization problems. *Frontiers in Quantum Science and Technology*. 2023, volume 2. ISSN 2813-2181. Available from: https://doi.org/10.3389/frqst.2023.1128576.

[AHD+11] R. Armstrong, B. J. Hall, J. Doyle, E. Waters. 'Scoping the scope' of a cochrane review. *Journal of Public Health*. 2011, volume 33, number 1, pp. 147–150. ISSN 1741-3842. Available from: https://doi.org/10.1093/pubmed/fdr015. _eprint: https://academic.oup.com/jpubhealth/article-pdf/33/1/147/4426880/fdr015.pdf.

[Ale19] G. Aleksandrowicz. *Qiskit: An Open-source Framework for Quantum Computing*'. *Zenodo,* [doi: 10.5281/zenodo.2562111]. 2019.

[AV24] A. Orenstein, V. Chaudhary. Quantum Circuit Mapping Using Binary Integer Nonlinear Programming. In: *2024 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 2024, pp. 1105–1114. Available from: https://doi.org/10.1109/IPDPSW63119.2024.00182. Journal Abbreviation: 2024 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW).

[BBC+24] K. Blekos, D. Brand, A. Ceschini, C.-H. Chou, R.-H. Li, K. Pandya, A. Summer. A review on Quantum Approximate Optimization Algorithm and its variants. *Physics Reports*. 2024, volume 1068, pp. 1–66. ISSN 0370-1573. Available from: https://doi.org/10.1016/j.physrep.2024.03.002.

[BDS25] S. Bhattacharjee, K. Das, B. Sarkar. PSO inspired global neighbourhood based Qubit mapping: a new approach. *EUROPEAN PHYSICAL JOURNAL PLUS*. 2025, volume 140, number 1. ISSN 2190-5444. Available from: https://doi.org/10.1140/epjp/s13360-024-05928-z.

[BM24] T. G. de Brugière, S. Martiel. *Faster and shorter synthesis of Hamiltonian simulation circuits*. 2024.

[CCC+20] J. Cao, R. J. Cogdell, D. F. Coker, H.-G. Duan, et al. Quantum biology revisited. *Science Advances*. 2020, volume 6, number 14, eaaz4888.

[CK19] J. Choi, J. Kim. A tutorial on quantum approximate optimization algorithm (QAOA): Fundamentals and applications. In: *2019 international conference on information and communication technology convergence (ICTC)*. IEEE, 2019, pp. 138–142.

[CRO+19] Y. Cao, J. Romero, J. P. Olson, M. Degroote, et al. Quantum chemistry in the age of quantum computing. *Chemical reviews*. 2019, volume 119, number 19, pp. 10856–10915.

[CSS+25] C. Luo, S. Cao, S. Guo, C. Hu. Towards Effective Local Search for Qubit Mapping. *IEEE Transactions on Computers*. 2025, pp. 1–14. ISSN 1557-9956. Available from: https://doi.org/10.1109/TC.2025.3544869.

[CWL+24] H. Cao, Q. Wang, X. Li, M. Najafi, K. C.-C. Chang, R. Cheng. Large Subgraph Matching: A Comprehensive and Efficient Approach for Heterogeneous Graphs. In: *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. 2024, pp. 2972–2985. Available from: https://doi.org/10.1109/ICDE60146.2024.00231.

[CYK+24] C.-Y. Cheng, C.-Y. Yang, Y.-H. Kuo, R.-C. Wang, H.-C. Cheng, C.-Y. (Huang. Robust Qubit Mapping Algorithm via Double-Source Optimal Routing on Large Quantum Circuits. *ACM Transactions on Quantum Computing*. 2024, volume 5, number 3. Available from: https://doi.org/10.1145/3680291. Place: New York, NY, USA Publisher: Association for Computing Machinery.

[DKM+25] A. Dubal, D. Kremer, S. Martiel, V. Villar, D. Wang, J. Cruz-Benito. *Pauli Network Circuit Synthesis with Reinforcement Learning*. 2025. Available also from: https://arxiv.org/abs/2503.14448.

[DZL20] H. Deng, Y. Zhang, Q. Li. Codar: A Contextual Duration-Aware Qubit Mapping for Various NISQ Devices. In: *2020 57th ACM/IEEE Design Automation Conference (DAC)*. 2020, pp. 1–6. Available from: https://doi.org/10.1109/DAC18072.2020.9218561.

[FGG14a] E. Farhi, J. Goldstone, S. Gutmann. *A Quantum Approximate Optimization Algorithm*. 2014. Available also from: https://arxiv.org/abs/1411.4028.

[FGG14b] E. Farhi, J. Goldstone, S. Gutmann. A quantum approximate optimization algorithm. *arXiv preprint arXiv:1411.4028*. 2014.

[Fis36] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*. 1936, volume 7, number 2, pp. 179–188.

[FMB24] F. Fuerrutter, G. Munoz-Gil, H. J. Briegel. Quantum circuit synthesis with diffusion models. *NATURE MACHINE INTELLIGENCE*. 2024, volume 6, number 5. Available from: https://doi.org/10.1038/s42256-024-00831-9.

[FMM+12] A. G. Fowler, M. Mariantoni, J. M. Martinis, A. N. Cleland. Surface codes: Towards practical large-scale quantum computation. *Phys. Rev. A*. 2012, volume 86, p. 032324. Available from: https://doi.org/10.1103/PhysRevA.86.032324.

[Fuj25] Fujitsu. *Digital Annealer – Quantum-Inspired Computing Technology, Available Today* [https://www.fujitsu.com/global/services/business-services/digital-annealer/]. 2025. [Online; accessed 14 June 2025].

[FZW+23] H. Fu, M. Zhu, J. Wu, W. Xie, Z. Su, X. Li, IEEE. Effective and Efficient Qubit Mapper. In: *Chinese Academy of Sciences*. 2023. ISBN 1933-7760. Available from: https://doi.org/10.1109/ICCAD57390.2023.10323857.

[GJS76] M. Garey, D. Johnson, L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*. 1976, volume 1, number 3, pp. 237–267. ISSN 0304-3975. Available from: https://doi.org/https://doi.org/10.1016/0304-3975(76)90059-1.

[Got98] D. Gottesman. *The Heisenberg Representation of Quantum Computers*. 1998. Available also from: https://arxiv.org/abs/quant-ph/9807006.

[HBM+21] H.-Y. Huang, M. Broughton, M. Mohseni, R. Babbush, S. Boixo, H. Neven, J. R. McClean. Power of data in quantum machine learning. *Nature Communications*. 2021, volume 12, p. 2631. Available from: https://doi.org/10.1038/s41467-021-22539-9.

[HGL+23] D. Herman, C. Googin, X. Liu, Y. Sun, A. Galda, I. Safro, M. Pistoia, Y. Alexeev. Quantum computing for finance. *Nature Reviews Physics*. 2023, volume 5, number 8, pp. 450–465.

[IKK+25] T. Ito, N. Kakimura, N. Kamiyama, Y. Kobayashi, Y. Okamoto. Algorithmic Theory of Qubit Routing in the Linear Nearest Neighbor Architectures. *ACM Transactions on Quantum Computing*. 2025. Available from: https://doi.org/10.1145/3722119. Place: New York, NY, USA Publisher: Association for Computing Machinery.

[JDX24] H. Jiang, Y. Deng, M. Xu. Qubit Mapping Based on Tabu Search. *JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY*. 2024, volume 39, number 2, pp. 421–433. ISSN 1000-9000. Available from: https://doi.org/10.1007/s11390-023-2121-5.

[JFD+25] H. Jiang, J. Fu, Y. Deng, J. Wu. A binary integer programming-based method for qubit mapping in sparse architectures. *ACTA INFORMATICA*. 2025, volume 62, number 1. ISSN 0001-5903. Available from: https://doi.org/10.1007/s00236-024-00471-x.

[JKP24a] Y. Ji, K. F. Koenig, I. Polian. Improving the performance of digitized counterdiabatic quantum optimization via algorithm-oriented qubit mapping. *Physical Review A*. 2024, volume 110, number 3, p. 032421.

[JKP24b] Y. Ji, K. Koenig, I. Polian. Improving the performance of digitized counterdiabatic quantum optimization via algorithm-oriented qubit mapping. *PHYSICAL REVIEW A*. 2024, volume 110, number 3. ISSN 2469-9926. Available from: https://doi.org/10.1103/PhysRevA.110.032421.

[KC07] B. Kitchenham, S. Charters. Guidelines for performing Systematic Literature Reviews in Software Engineering. 2007, volume 2.

[KDS+24] A. Kole, K. Datta, I. Sengupta, R. Drechsler. Exploiting the Extended Neighborhood of Hexagonal Qubit Architecture for Mapping Quantum Circuits. *J. Emerg. Technol. Comput. Syst.* 2024, volume 20, number 4. ISSN 1550-4832. Available from: https://doi.org/10.1145/3688391. Place: New York, NY, USA Publisher: Association for Computing Machinery.

[KP26] G. Keibas, L. Petkevičius. *QuMeld: A Modular Framework for Benchmarking Qubit Mapping Algorithms*. 2026. Available also from: https://arxiv.org/abs/2603.01578.

[KSZ+23] S. Khadirsharbiyani, M. Sadeghi, M. Zarch, J. Kotra, M. Kandemir. TRIM: crossTalk-awaRe qubIt Mapping for multiprogrammed quantum systems. In: S. Ali, C. Ardagna, N. Atukorala, J. Barzen, et al. (editors). *Pennsylvania Commonwealth System of Higher Education (PCSHE)*. 2023, pp. 138–148. ISBN 979-8-3503-0479-4. Available from: https://doi.org/10.1109/QSW59989.2023.00025.

[KVP+25] D. Kremer, V. Villar, H. Paik, I. Duran, I. Faro, J. Cruz-Benito. *Practical and efficient quantum circuit synthesis and transpiling with Reinforcement Learning*. 2025. Available also from: https://arxiv.org/abs/2405.13196.

[LD24] L. Liu, X. Dou. QuCloud+: A Holistic Qubit Mapping Scheme for Single/Multi-programming on 2D/3D NISQ Quantum Computers. *ACM Trans. Archit. Code Optim.* 2024, volume 21, number 1. ISSN 1544-3566. Available from: https://doi.org/10.1145/3631525. Place: New York, NY, USA Publisher: Association for Computing Machinery.

[LDX19a] G. Li, Y. Ding, Y. Xie. Tackling the Qubit Mapping Problem for NISQ-Era Quantum Devices. In: *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. Providence, RI, USA: Association for Computing Machinery, 2019, pp. 1001–1014. ASPLOS '19. ISBN 9781450362405. Available from: https://doi.org/10.1145/3297858.3304023.

[LDX19b] G. Li, Y. Ding, Y. Xie. Tackling the qubit mapping problem for NISQ-era quantum devices. In: *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. 2019, pp. 1001–1014.

[LG20] J. Li, S. Ghosh. Quantum-soft qubo suppression for accurate object detection. In: *European conference on computer vision*. Springer, 2020, pp. 158–173.

[LLW+24] T. Li, W. Liu, F. Wang, IEEE. Optimization of Qubit Mapping Model Based on Deep Q Network. In: *Zhengzhou University*. 2024, pp. 538–543. ISBN 979-8-3503-8678-3. Available from: https://doi.org/10.1109/ICCEA62105.2024.10603880.

[LNC+23] S. Li, K. Nguyen, Z. Clare, Y. Feng, IEEE. Single-Qubit Gates Matter for Optimising Quantum Circuit Depth in Qubit Mapping. In: *University of Technology Sydney*. 2023. ISBN 1933-7760. Available from: https://doi.org/10.1109/ICCAD57390.2023.10323863.

[LSK+22] A. Li, S. Stein, S. Krishnamoorthy, J. Ang. *QASMBench: A Low-level QASM Benchmark Suite for NISQ Evaluation and Simulation*. 2022. Available also from: https://arxiv.org/abs/2005.13018.

[LW24] Y. Lou, C. Wang. A Generalized Community-Structure-Aware Optimization Framework for Efficient Subgraph Matching in Social Network Analysis. *IEEE TRANSACTIONS ON COMPUTATIONAL SOCIAL SYSTEMS*. 2024, volume 11, number 2, pp. 2545–2557. ISSN 2329-924X. Available from: https://doi.org/10.1109/TCSS.2023.3303476.

[LZZ+24] H. Liu, B. Zhang, Y. Zhu, H. Yang, B. Zhao. QM-DLA: an efficient qubit mapping method based on dynamic look-ahead strategy. *SCIENTIFIC REPORTS*. 2024, volume 14, number 1. ISSN 2045-2322. Available from: https://doi.org/10.1038/s41598-024-64061-0.

[MK13] C. Monroe, J. Kim. Scaling the ion trap quantum processor. *Science*. 2013, volume 339, number 6124, pp. 1164–1169.

[MMJ+22] H. Mustafa, S. N. Morapakula, P. Jain, S. Ganguly. Variational quantum algorithms for chemical simulation and drug discovery. In: *2022 International Conference on Trends in Quantum Computing and Emerging Business Technologies (TQCEBT)*. IEEE, 2022, pp. 1–8.

[MWB+22] C. Moussa, H. Wang, T. Bäck, V. Dunjko. Unsupervised strategies for identifying optimal parameters in quantum approximate optimization algorithm. *EPJ Quantum Technology*. 2022, volume 9, number 1, p. 11.

[MZ23] N. Mariella, S. Zhuk. A doubly stochastic matrices-based approach to optimal qubit routing. *QUANTUM INFORMATION PROCESSING*. 2023, volume 22, number 7. ISSN 1570-0755. Available from: https://doi.org/10.1007/s11128-023-04023-z.

[NC10] M. A. Nielsen, I. L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.

[NT21] S. Niu, A. Todri-Sanial. Analyzing crosstalk error in the NISQ era. In: *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2021, pp. 428–430.

[NT23] P. Nation, M. Treinish. Suppressing Quantum Circuit Errors Due to System Variability. *PRX QUANTUM*. 2023, volume 4, number 1. ISSN 2691-3399. Available from: https://doi.org/10.1103/PRXQuantum.4.010327.

[Pap26] L. Papp. *CLARYON: Classical-quantum AI for Reproducible Explainable Open-source Medicine*. GitHub, 2026. Available also from: https://github.com/lpapp-muw/CLARYON.

[PBW23] T. Peham, L. Burgholzer, R. Wille. On Optimal Subarchitectures for Quantum Circuit Mapping. *ACM Transactions on Quantum Computing*. 2023, volume 4, number 4. Available from: https://doi.org/10.1145/3593594. Place: New York, NY, USA Publisher: Association for Computing Machinery.

[PDB+24] A. Paradis, J. Dekoninck, B. Bichsel, M. Vechev. Synthetiq: Fast and Versatile Quantum Circuit Synthesis. *Proc. ACM Program. Lang.* 2024, volume 8, number OOPSLA1. Available from: https://doi.org/10.1145/3649813.

[PMS+14] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik, J. L. O'brien. A variational eigenvalue solver on a photonic quantum processor. *Nature communications*. 2014, volume 5, number 1, p. 4213.

[Pow94] M. J. D. Powell. A Direct Search Optimization Method That Models the Objective and Constraint Functions by Linear Interpolation. In: 1994. Available from: https://doi.org/10.1007/978-94-015-8330-5_4.

[Pre18] J. Preskill. Quantum Computing in the NISQ era and beyond. *Quantum*. 2018, volume 2, p. 79. ISSN 2521-327X. Available from: https://doi.org/10.22331/q-2018-08-06-79.

[PSK24] R. Pfister, G. Schubert, M. Kröll. Transfer of Logistics Optimizations to Material Flow Resource Optimizations using Quantum Computing. *Procedia Computer Science*. 2024, volume 232, pp. 32–42.

[PVS+26] L. Papp, D. Visvikis, M. Sollini, K. Shi, M. Kirienko. The Dawn of Quantum AI in Nuclear Medicine: an EANM Perspective. *The EANM Journal*. 2026. In revision.

[RMX+20] Y. Ruan, S. Marsh, X. Xue, Z. Liu, J. Wang, et al. The quantum approximate algorithm for solving traveling salesman problem. *Computers, Materials and Continua*. 2020, volume 63, number 3, pp. 1237–1247.

[SAK25] M. Shirotani, M. Amagasaki, M. Kiyama. ENDNet: Extra-Node Decision Network for Subgraph Matching. *IEEE Access*. 2025, volume 13, pp. 32424–32433. Available from: https://doi.org/10.1109/ACCESS.2025.3543206.

[SIM+23] S. Khandavilli, I. Palanisamy, M. V. Nguyen, T. V. Le, T. N. Nguyen, T. N. Dinh. Towards Fidelity-Optimal Qubit Mapping on NISQ Computers. In: *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*. 2023, volume 01, pp. 89–98. Available from: https://doi.org/10.1109/QCE57702.2023.00019. Journal Abbreviation: 2023 IEEE International Conference on Quantum Computing and Engineering (QCE).

[SM91] K. Shahookar, P. Mazumder. VLSI cell placement techniques. *ACM Comput. Surv.* 1991, volume 23, number 2, pp. 143–220. ISSN 0360-0300. Available from: https://doi.org/10.1145/103724.103725.

[SMM+23] S. Khadirsharbiyani, M. Sadeghi, M. E. Zarch, J. Kotra, M. T. Kandemir. TRIM: crossTalk-awaRe qubIt Mapping for multiprogrammed quantum systems. In: *2023 IEEE International Conference on Quantum Software (QSW)*. 2023, pp. 138–148. Available from: https://doi.org/10.1109/QSW59989.2023.00025. Journal Abbreviation: 2023 IEEE International Conference on Quantum Software (QSW).

[SW22] R. Shaydulin, S. M. Wild. Importance of kernel bandwidth in quantum machine learning. *Physical Review A*. 2022, volume 106, number 4, p. 042407. Available from: https://doi.org/10.1103/PhysRevA.106.042407.

[TCJ24] W.-H. Tseng, Y.-W. Chang, J.-H. R. Jiang. Satisfiability Modulo Theories-Based Qubit Mapping for Trapped-Ion Quantum Computing Systems. In: *Proceedings of the 2024 International Symposium on Physical Design*. New York, NY, USA: Association for Computing Machinery, 2024, pp. 245–253. ISPD '24. ISBN 979-8-4007-0417-8. Available from: https://doi.org/10.1145/3626184.3633329. event-place: Taipei, Taiwan.

[tea20] T. pandas development team. *pandas-dev/pandas: Pandas*. Zenodo, 2020. Latest. Available from: https://doi.org/10.5281/zenodo.3509134.

[TFX+23] T. LeCompte, F. Qi, X. Yuan, N. -F. Tzeng, M. H. Najafi, L. Peng. Machine-Learning-Based Qubit Allocation for Error Reduction in Quantum Circuits. *IEEE Transactions on Quantum Engineering*. 2023, volume 4, pp. 1–14. ISSN 2689-1808. Available from: https://doi.org/10.1109/TQE.2023.3301899.

[TSL23] Y. Tang, T. Shang, J. Liu. Dynamic full quantum one-way function based on quantum circuit mapping. *QUANTUM INFORMATION PROCESSING*. 2023, volume 22, number 8. ISSN 1570-0755. Available from: https://doi.org/10.1007/s11128-023-04065-3.

[TWC+24] S. Thanasilp, S. Wang, M. Cerezo, Z. Holmes. Exponential concentration in quantum kernel methods. *Nature Communications*. 2024, volume 15, p. 5200. Available from: https://doi.org/10.1038/s41467-024-49287-w.

[UKB20] A. Uvarov, A. Kardashin, J. D. Biamonte. Machine learning phase transitions with a quantum processor. *Physical Review A*. 2020, volume 102, number 1, p. 012415.

[UL23] H. Ushijima Mwesigwa, X. Liu. An Ising-based Model for Qubit Mapping. In: *Proceedings of the SC '23 Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis*. New York, NY, USA: Association for Computing Machinery, 2023, pp. 1492–1498. SC-W '23. ISBN 979-8-4007-0785-8. Available from: https://doi.org/10.1145/3624062.3624225. event-place: Denver, CO, USA.

[WB23] R. Wille, L. Burgholzer. MQT QMAP: Efficient Quantum Circuit Mapping. In: *Proceedings of the 2023 International Symposium on Physical Design*. New York, NY, USA: Association for Computing Machinery, 2023, pp. 198–204. ISPD '23. ISBN 978-1-4503-9978-4. Available from: https://doi.org/10.1145/3569052.3578928. event-place: Virtual Event, USA.

[WBC+21] Y. Wu, W.-S. Bao, S. Cao, F. Chen, et al. Strong quantum computational advantage using a superconducting quantum processor. *Physical review letters*. 2021, volume 127, number 18, p. 180501.

[WDE+23] K. Wintersperger, F. Dommert, T. Ehmer, A. Hoursanov, et al. Neutral atom quantum computing hardware: performance and end-user perspective. *EPJ Quantum Technology*. 2023, volume 10, number 1, p. 32.

[WSM94] W. H. Wolberg, W. N. Street, O. L. Mangasarian. Machine learning techniques to diagnose breast cancer from image-processed nuclear features of fine needle aspirates. *Cancer Letters*. 1994, volume 77, number 2–3, pp. 163–171.

[XZZ21] L. Xie, J. Zhai, W. Zheng. Mitigating Crosstalk in Quantum Computers through Commutativity-Based Instruction Reordering. In: *2021 58th ACM/IEEE Design Automation Conference (DAC)*. 2021, pp. 445–450. Available from: https://doi.org/10.1109/DAC18074.2021.9586145.

[YLJ24] Y. Huang, L. Wang, J. Xu. Quantum Entanglement Path Selection and Qubit Allocation via Adversarial Group Neural Bandits. *IEEE Transactions on Networking*. 2024, pp. 1–12. ISSN 2998-4157. Available from: https://doi.org/10.1109/TNET.2024.3510550.

[YYH+24] Y. Li, Y. Zhang, H. Deng, M. Chen, Z. Li. PauliForest: Connectivity-Aware Synthesis and Pauli-Oriented Qubit Mapping for Near Term Quantum Simulation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*. 2024, pp. 1–1. ISSN 1937-4151. Available from: https://doi.org/10.1109/TCAD.2024.3509794.

[ZHL24] S. Zhang, K. Huang, L. Li. Depth-optimized quantum circuit synthesis for diagonal unitary operators with asymptotically optimal gate count. *PHYSICAL REVIEW A*. 2024, volume 109, number 4. ISSN 2469-9926. Available from: https://doi.org/10.1103/PhysRevA.109.042601.

[ZTH+24] H. Zou, M. Treinish, K. Hartman, A. Ivrii, J. Lishman. *LightSABRE: A Lightweight and Enhanced SABRE Algorithm*. 2024. Available also from: https://arxiv.org/abs/2409.08368.

[ZWY+25] C. Zhu, X. Wu, Z. Yang, J. Wang, A. Wu, S. Zheng, X. Wang. *Quantum Compiler Design for Qubit Mapping and Routing: A Cross-Architectural Survey of Superconducting, Trapped-Ion, and Neutral Atom Systems*. 2025. Available also from: https://arxiv.org/abs/2505.16891.

Appendixes

Appendix 1
Full results tables

Table 27. Full results for VQE H_2 Hamiltonian

Topology	Optimizer	SWAP	CNOT	Depth	Time (ms)
Google Willow Q105	doustra	30	126	65	77,77
	lightSABRE_decay	3	37	85	33,31
	lightSABRE_lookahead	3	35	85	98,52
	pauliforest	6	40	38	450,20
	qiskit_ai	8	60	192	104,08
	rustiq	3	39	87	32,77
Hexagonal Lattice Q54	doustra	43	160	80	58,23
	lightSABRE_decay	3	35	129	36,02
	lightSABRE_lookahead	3	36	129	120,51
	pauliforest	6	40	38	80,07
	qiskit_ai	10	66	186	95,37
	rustiq	4	40	114	25,96
Ibm Almaden Q20	doustra	17	74	73	42,44
	lightSABRE_decay	4	40	126	25,38
	lightSABRE_lookahead	4	43	130	113,08
	pauliforest	6	40	38	22,24
	qiskit_ai	6	54	225	96,97
	rustiq	4	42	130	25,39
Ibm Cambridge Q28	doustra	18	80	67	43,32
	lightSABRE_decay	5	49	71	13,26
	lightSABRE_lookahead	5	49	71	113,23
	pauliforest	6	40	38	28,02
	qiskit_ai	6	54	225	94,22
	rustiq	5	49	71	12,64
Ibm Eagle Q127	doustra	18	79	67	62,76
	lightSABRE_decay	5	51	76	21,54
	lightSABRE_lookahead	5	49	71	141,78
	pauliforest	6	40	38	765,43
	qiskit_ai	6	54	221	108,51
	rustiq	5	51	76	20,10
Ibm Falcon Q27	doustra	18	80	73	41,95
	lightSABRE_decay	4	42	119	28,73
	lightSABRE_lookahead	4	39	119	128,48
	pauliforest	6	40	38	26,59
	qiskit_ai	6	54	225	95,61
	rustiq	4	42	120	28,52

Continued on next page

Table 27 – continued from previous page

Topology	Optimizer	SWAP	CNOT	Depth	Time (ms)
Ibm Heron Q133	doustra	17	74	67	66,44
	lightSABRE_decay	5	51	76	22,13
	lightSABRE_lookahead	5	51	76	132,54
	pauliforest	6	40	38	874,40
	qiskit_ai	6	54	97	116,83
	rustiq	5	51	76	20,83
Ibm Manhattan Q65	doustra	17	74	73	49,92
	lightSABRE_decay	3	33	148	57,01
	lightSABRE_lookahead	3	34	151	170,17
	pauliforest	6	40	38	123,23
	qiskit_ai	6	54	221	99,63
	rustiq	3	36	149	53,63
Ibm Montreal Q27	doustra	47	191	100	142,24
	lightSABRE_decay	4	43	118	23,30
	lightSABRE_lookahead	4	43	118	130,76
	pauliforest	6	40	38	26,20
	qiskit_ai	19	93	155	99,53
	rustiq	4	41	114	23,09
Ibm Paughkeepsie Q20	doustra	18	82	67	41,58
	lightSABRE_decay	4	42	129	26,58
	lightSABRE_lookahead	4	42	128	117,94
	pauliforest	6	40	38	21,58
	qiskit_ai	6	54	225	95,20
	rustiq	4	43	125	24,39
Ibm Reuschlikon Q16	doustra	18	86	54	45,10
	lightSABRE_decay	3	39	97	21,30
	lightSABRE_lookahead	3	39	97	104,68
	pauliforest	6	40	38	20,35
	qiskit_ai	6	54	196	91,26
	rustiq	3	39	107	20,42
Ibm Rochester Q53	doustra	13	63	61	48,08
	lightSABRE_decay	5	51	76	15,96
	lightSABRE_lookahead	5	49	71	134,11
	pauliforest	6	40	38	75,98
	qiskit_ai	6	54	225	100,49
	rustiq	5	49	71	13,94
Ibm Tokyo Q20	doustra	12	71	56	53,32
	lightSABRE_decay	0	30	81	21,46
	lightSABRE_lookahead	0	30	109	78,49
	pauliforest	0	22	30	19,68
	qiskit_ai	8	60	184	90,58
	rustiq	0	30	81	19,86

Continued on next page

Table 27 – continued from previous page

Topology	Optimizer	SWAP	CNOT	Depth	Time (ms)
Ionq Harmony Q9	doustra	0	23	36	37,41
	lightSABRE_decay	0	36	55	49,50
	lightSABRE_lookahead	0	36	55	63,17
	pauliforest	0	22	30	17,25
	qiskit_ai	0	36	70	88,01
	rustiq	0	36	55	48,93
Rigetti Novera Q9	doustra	16	73	58	45,49
	lightSABRE_decay	1	31	101	35,19
	lightSABRE_lookahead	1	34	100	90,61
	pauliforest	6	40	38	19,74
	qiskit_ai	7	57	208	102,67
	rustiq	3	37	93	27,79
Riken Fujitsu Q256	doustra	18	77	61	99,14
	lightSABRE_decay	3	41	111	48,56
	lightSABRE_lookahead	3	39	106	121,30
	pauliforest	6	40	38	6058,29
	qiskit_ai	6	54	225	152,21
	rustiq	3	39	86	48,95

Table 28. Full results for random Pauli strings hamiltonian

Topology	Optimizer	SWAP	CNOT	Depth	Time (ms)
Google Willow Q105	doustra	175	590	190	702,53
	lightSABRE_decay	6	74	127	34,20
	lightSABRE_lookahead	6	74	128	221,42
	pauliforest	11	78	46	468,28
	qiskit_ai	40	180	316	141,43
	rustiq	7	77	127	40,20
Hexagonal Lattice Q54	doustra	179	613	215	635,42
	lightSABRE_decay	13	91	198	34,43
	lightSABRE_lookahead	11	84	173	264,59
	pauliforest	18	99	52	96,08
	qiskit_ai	64	252	363	143,66
	rustiq	12	91	171	33,74
Ibm Almaden Q20	doustra	112	402	204	193,97
	lightSABRE_decay	13	91	191	27,67
	lightSABRE_lookahead	11	86	169	280,32
	pauliforest	18	99	53	38,75
	qiskit_ai	40	180	381	126,02
	rustiq	12	88	182	27,99

Continued on next page

Table 28 – continued from previous page

Topology	Optimizer	SWAP	CNOT	Depth	Time (ms)
Ibm Cambridge Q28	doustra	208	691	227	448,33
	lightSABRE_decay	13	94	136	22,63
	lightSABRE_lookahead	12	93	147	262,40
	pauliforest	31	138	63	45,64
	qiskit_ai	64	252	326	138,19
	rustiq	13	93	136	21,27
Ibm Eagle Q127	doustra	184	615	228	250,19
	lightSABRE_decay	12	87	181	40,39
	lightSABRE_lookahead	12	87	177	276,67
	pauliforest	31	138	62	789,47
	qiskit_ai	27	141	451	148,02
	rustiq	12	87	177	40,13
Ibm Falcon Q27	doustra	206	687	260	490,76
	lightSABRE_decay	12	93	107	15,64
	lightSABRE_lookahead	12	93	107	256,16
	pauliforest	31	138	62	43,27
	qiskit_ai	68	264	382	149,33
	rustiq	13	96	101	17,33
Ibm Heron Q133	doustra	177	580	214	214,92
	lightSABRE_decay	12	93	148	29,94
	lightSABRE_lookahead	12	93	148	281,20
	pauliforest	31	138	62	894,78
	qiskit_ai	24	132	450	170,10
	rustiq	12	92	148	31,06
Ibm Manhattan Q65	doustra	189	635	221	216,92
	lightSABRE_decay	12	93	110	20,29
	lightSABRE_lookahead	12	93	110	257,45
	pauliforest	31	138	61	140,65
	qiskit_ai	29	147	453	136,70
	rustiq	13	92	147	31,08
Ibm Montreal Q27	doustra	177	613	257	308,54
	lightSABRE_decay	12	93	114	17,39
	lightSABRE_lookahead	12	93	110	251,72
	pauliforest	31	138	63	43,53
	qiskit_ai	62	246	412	135,75
	rustiq	12	93	114	17,39
Ibm Paughkeepsie Q20	doustra	160	543	216	196,50
	lightSABRE_decay	13	94	215	26,08
	lightSABRE_lookahead	11	86	189	261,01
	pauliforest	20	105	55	37,32
	qiskit_ai	50	210	366	130,92
	rustiq	13	93	187	23,52

Continued on next page

Table 28 – continued from previous page

Topology	Optimizer	SWAP	CNOT	Depth	Time (ms)
Ibm Reuschlikon Q16	doustra	92	336	162	182,26
	lightSABRE_decay	7	77	125	303,38
	lightSABRE_lookahead	6	69	163	255,70
	pauliforest	14	87	50	45,40
	qiskit_ai	26	138	438	136,65
	rustiq	6	73	135	21,43
Ibm Rochester Q53	doustra	218	714	244	507,06
	lightSABRE_decay	12	86	182	33,87
	lightSABRE_lookahead	12	87	185	282,36
	pauliforest	31	138	61	94,59
	qiskit_ai	61	243	350	157,04
	rustiq	12	88	176	32,05
Ibm Tokyo Q20	doustra	65	266	149	185,27
	lightSABRE_decay	3	64	145	21,11
	lightSABRE_lookahead	2	60	131	150,67
	pauliforest	7	66	50	37,23
	qiskit_ai	33	159	377	121,00
	rustiq	3	62	142	19,78
Ionq Harmony Q9	doustra	0	61	82	154,66
	lightSABRE_decay	0	57	80	86,45
	lightSABRE_lookahead	0	57	77	99,76
	pauliforest	0	45	38	31,91
	qiskit_ai	0	60	94	100,79
	rustiq	0	57	77	85,84
Rigetti Novera Q9	doustra	77	290	151	206,06
	lightSABRE_decay	8	81	138	21,97
	lightSABRE_lookahead	8	81	138	165,32
	pauliforest	10	75	46	34,79
	qiskit_ai	33	159	335	129,89
	rustiq	8	81	137	19,60
Riken Fujitsu Q256	doustra	137	460	173	385,36
	lightSABRE_decay	8	77	169	65,75
	lightSABRE_lookahead	6	70	155	275,47
	pauliforest	11	78	47	6040,20
	qiskit_ai	25	135	451	179,00
	rustiq	7	74	168	81,00

Table 29. Full results for VQE LiH Hamiltonian algorithm

Topology	Optimizer	SWAP	CNOT	Depth	Time (ms)
Google Willow Q105	doustra	3277	11373	4756	9903,85
	lightSABRE_decay	1037	8242	14379	1309,12
	lightSABRE_lookahead	968	8026	13853	19659,53
	pauliforest	1635	7247	3334	1564,57
	qiskit_ai	3921	17824	32004	6422,29
	rustiq	1016	8119	14533	1389,97
Hexagonal Lattice Q54	doustra	4488	15070	5755	9204,05
	lightSABRE_decay	1646	10077	16734	1465,27
	lightSABRE_lookahead	1556	9711	16535	27571,49
	pauliforest	2264	9137	3564	1269,75
	qiskit_ai	4753	20320	32132	6753,65
	rustiq	1605	9994	15141	1439,77
Ibm Almaden Q20	doustra	3677	12591	5612	8311,81
	lightSABRE_decay	1614	9925	15752	1498,14
	lightSABRE_lookahead	1562	9858	14409	27149,52
	pauliforest	2279	9184	3568	1198,59
	qiskit_ai	4801	20463	32677	6670,51
	rustiq	1612	9948	16652	1533,46
Ibm Cambridge Q28	doustra	5231	17268	6484	11026,39
	lightSABRE_decay	2025	11294	13941	1353,81
	lightSABRE_lookahead	1987	11019	14277	28194,06
	pauliforest	3058	11518	3969	1293,25
	qiskit_ai	5801	23463	31655	7159,70
	rustiq	2027	11215	14153	1481,42
Ibm Eagle Q127	doustra	5293	17435	6552	11050,04
	lightSABRE_decay	1987	10985	17872	1727,02
	lightSABRE_lookahead	1942	10849	17552	28680,15
	pauliforest	3087	11604	3988	2072,82
	qiskit_ai	6481	25503	33628	7592,73
	rustiq	2026	11158	14831	1562,50
Ibm Falcon Q27	doustra	4916	16251	6413	9825,50
	lightSABRE_decay	2018	11049	14963	1698,85
	lightSABRE_lookahead	1986	11022	14303	27704,95
	pauliforest	3095	11630	3976	1289,79
	qiskit_ai	5732	23256	31575	7266,23
	rustiq	2022	11157	14420	1515,18
Ibm Heron Q133	doustra	5485	18021	6601	11722,74
	lightSABRE_decay	2021	11229	13463	1483,93
	lightSABRE_lookahead	1905	10508	17656	29099,08
	pauliforest	3072	11561	4002	2173,86
	qiskit_ai	6352	25116	34087	7537,94
	rustiq	2022	11149	14825	1581,02

Continued on next page

Table 29 – continued from previous page

Topology	Optimizer	SWAP	CNOT	Depth	Time (ms)
Ibm Manhattan Q65	doustra	5393	17748	6568	10200,66
	lightSABRE_decay	2022	11265	13414	1338,37
	lightSABRE_lookahead	1898	11091	10560	27151,23
	pauliforest	3076	11572	3987	1380,18
	qiskit_ai	6088	24324	33209	7397,71
	rustiq	2013	11123	14410	1505,92
Ibm Montreal Q27	doustra	5081	16835	6573	8997,51
	lightSABRE_decay	2018	11035	14963	1695,29
	lightSABRE_lookahead	1994	11124	14208	27116,99
	pauliforest	3046	11480	3968	1284,40
	qiskit_ai	6033	24159	34272	7324,80
	rustiq	2027	11171	14247	1506,93
Ibm Paughkeepsie Q20	doustra	4569	15303	6259	8961,75
	lightSABRE_decay	1598	9879	13638	1429,24
	lightSABRE_lookahead	1516	9757	13538	26523,55
	pauliforest	2527	9925	3696	1228,71
	qiskit_ai	5027	21142	31933	6750,58
	rustiq	1589	10020	13760	1329,07
Ibm Reuschlikon Q16	doustra	2731	9766	4920	8953,89
	lightSABRE_decay	1129	8501	13852	1617,31
	lightSABRE_lookahead	1046	8191	14686	20673,83
	pauliforest	2206	8960	3502	1179,73
	qiskit_ai	4113	18400	31329	6391,52
	rustiq	1108	8421	14286	1366,53
Ibm Rochester Q53	doustra	5346	17621	6537	10685,83
	lightSABRE_decay	2021	11073	15327	1706,76
	lightSABRE_lookahead	1907	10840	16756	27767,64
	pauliforest	3077	11573	3991	1335,50
	qiskit_ai	5790	23431	32517	7258,94
	rustiq	2012	11083	15669	1593,75
Ibm Tokyo Q20	doustra	2083	7806	3988	7690,61
	lightSABRE_decay	556	6824	13764	1194,95
	lightSABRE_lookahead	416	6334	14257	15975,40
	pauliforest	1127	5722	3136	1070,47
	qiskit_ai	2385	13216	30989	5694,87
	rustiq	534	6624	14554	1287,58
Ionq Harmony Q9	doustra	-1	-1	-1	0,00
	lightSABRE_decay	-1	-1	-1	0,00
	lightSABRE_lookahead	-1	-1	-1	0,00
	pauliforest	-1	-1	-1	0,00
	qiskit_ai	-1	-1	-1	0,00
	rustiq	-1	-1	-1	0,00

Continued on next page

Table 29 – continued from previous page

Topology	Optimizer	SWAP	CNOT	Depth	Time (ms)
Rigetti Novera Q9	doustra	-1	-1	-1	0,00
	lightSABRE_decay	-1	-1	-1	0,00
	lightSABRE_lookahead	-1	-1	-1	0,00
	pauliforest	-1	-1	-1	0,00
	qiskit_ai	-1	-1	-1	0,00
	rustiq	-1	-1	-1	0,00
Riken Fujitsu Q256	doustra	3330	11539	4728	13066,18
	lightSABRE_decay	994	8136	14362	1324,06
	lightSABRE_lookahead	953	7922	14940	20665,03
	pauliforest	1641	7262	3282	7116,02
	qiskit_ai	6417	25311	33853	7509,48
	rustiq	1027	8145	14576	1474,06

Table 30. Full results for QAOA Max Cut Hamiltonian algorithm

Topology	Optimizer	SWAP	CNOT	Depth	Time (ms)
Google Willow Q105	doustra	24	97	50	50,03
	lightSABRE_decay	1	11	33	28,46
	lightSABRE_lookahead	1	11	33	50,49
	pauliforest	6	38	31	447,23
	qiskit_ai	3	21	41	91,23
	rustiq	1	9	42	26,33
Hexagonal Lattice Q54	doustra	38	149	61	81,41
	lightSABRE_decay	2	16	23	13,10
	lightSABRE_lookahead	2	18	25	34,60
	pauliforest	9	47	30	77,64
	qiskit_ai	5	27	56	88,95
	rustiq	2	18	23	10,25
Ibm Almaden Q20	doustra	21	93	67	33,39
	lightSABRE_decay	2	14	38	14,22
	lightSABRE_lookahead	2	14	38	35,08
	pauliforest	9	47	31	18,79
	qiskit_ai	4	24	18	87,44
	rustiq	2	16	32	11,75
Ibm Cambridge Q28	doustra	17	75	56	32,86
	lightSABRE_decay	2	16	23	9,67
	lightSABRE_lookahead	2	16	23	32,97
	pauliforest	9	47	30	24,32
	qiskit_ai	4	24	17	81,60
	rustiq	2	18	25	9,41

Continued on next page

Table 30 – continued from previous page

Topology	Optimizer	SWAP	CNOT	Depth	Time (ms)
Ibm Eagle Q127	doustra	17	75	56	46,03
	lightSABRE_decay	2	16	29	18,00
	lightSABRE_lookahead	2	16	29	43,06
	pauliforest	9	47	31	762,45
	qiskit_ai	4	24	17	93,16
	rustiq	2	18	25	17,17
Ibm Falcon Q27	doustra	29	116	81	36,52
	lightSABRE_decay	2	16	34	12,13
	lightSABRE_lookahead	2	16	29	33,68
	pauliforest	9	47	31	22,98
	qiskit_ai	3	21	45	84,01
	rustiq	1	14	37	30,17
Ibm Heron Q133	doustra	21	93	71	48,48
	lightSABRE_decay	2	16	40	19,97
	lightSABRE_lookahead	2	16	40	44,92
	pauliforest	9	47	31	872,41
	qiskit_ai	4	24	18	98,72
	rustiq	2	14	41	20,17
Ibm Manhattan Q65	doustra	21	93	71	40,06
	lightSABRE_decay	2	14	37	15,42
	lightSABRE_lookahead	2	16	37	39,98
	pauliforest	9	47	31	120,08
	qiskit_ai	4	24	18	85,97
	rustiq	2	16	40	14,93
Ibm Montreal Q27	doustra	53	191	78	99,07
	lightSABRE_decay	2	16	23	10,09
	lightSABRE_lookahead	2	18	25	32,40
	pauliforest	9	47	30	23,87
	qiskit_ai	7	33	60	84,61
	rustiq	2	16	29	10,96
Ibm Paughkeepsie Q20	doustra	21	93	71	33,15
	lightSABRE_decay	1	11	38	34,06
	lightSABRE_lookahead	1	11	38	60,90
	pauliforest	9	47	31	19,01
	qiskit_ai	4	24	17	83,40
	rustiq	2	16	30	10,82
Ibm Reuschlikon Q16	doustra	20	90	61	33,69
	lightSABRE_decay	1	13	36	13,77
	lightSABRE_lookahead	1	11	36	31,38
	pauliforest	8	44	31	16,33
	qiskit_ai	4	24	33	83,50
	rustiq	1	13	32	10,27

Continued on next page

Table 30 – continued from previous page

Topology	Optimizer	SWAP	CNOT	Depth	Time (ms)
Ibm Rochester Q53	doustra	21	93	71	38,20
	lightSABRE_decay	2	14	37	16,93
	lightSABRE_lookahead	2	14	37	38,77
	pauliforest	9	47	30	73,07
	qiskit_ai	4	24	17	84,04
	rustiq	2	14	42	22,06
Ibm Tokyo Q20	doustra	13	72	45	57,72
	lightSABRE_decay	0	8	32	14,14
	lightSABRE_lookahead	0	8	26	30,54
	pauliforest	2	26	26	18,49
	qiskit_ai	5	27	34	83,40
	rustiq	0	8	32	13,06
Ionq Harmony Q9	doustra	0	28	27	33,47
	lightSABRE_decay	0	12	15	45,23
	lightSABRE_lookahead	0	12	15	54,16
	pauliforest	0	20	24	13,75
	qiskit_ai	0	12	15	83,47
	rustiq	0	12	15	43,26
Rigetti Novera Q9	doustra	18	82	52	35,31
	lightSABRE_decay	1	11	34	16,28
	lightSABRE_lookahead	1	11	33	33,33
	pauliforest	6	38	31	15,42
	qiskit_ai	2	18	32	86,04
	rustiq	1	11	42	15,39
Riken Fujitsu Q256	doustra	31	123	67	75,38
	lightSABRE_decay	1	11	43	53,37
	lightSABRE_lookahead	1	11	44	80,37
	pauliforest	6	38	31	6084,21
	qiskit_ai	4	24	18	129,15
	rustiq	1	9	39	55,51

Table 31. Full results for VQE EfficientSU2 ansatz

Topology	Optimizer	SWAP	CNOT	Depth	Time (ms)
Google Willow Q105	doustra	21	85	61	618,87
	lightSABRE_decay	0	2	22	32,47
	lightSABRE_lookahead	0	3	15	33,62
	pauliforest	-1	-1	-1	0,00
	qiskit_ai	3	13	30	93,43
	rustiq	0	2	23	31,33

Continued on next page

Table 31 – continued from previous page

Topology	Optimizer	SWAP	CNOT	Depth	Time (ms)
Hexagonal Lattice Q54	doustra	23	97	68	304,93
	lightSABRE_decay	0	3	18	10,99
	lightSABRE_lookahead	0	2	21	26,15
	pauliforest	-1	-1	-1	0,00
	qiskit_ai	6	22	37	89,18
	rustiq	0	3	17	10,24
Ibm Almaden Q20	doustra	16	64	54	191,31
	lightSABRE_decay	0	3	21	10,21
	lightSABRE_lookahead	0	4	12	19,08
	pauliforest	-1	-1	-1	0,00
	qiskit_ai	0	4	16	80,97
	rustiq	0	3	21	8,88
Ibm Cambridge Q28	doustra	16	64	54	212,20
	lightSABRE_decay	0	3	15	7,80
	lightSABRE_lookahead	0	3	17	23,23
	pauliforest	-1	-1	-1	0,00
	qiskit_ai	0	4	16	84,04
	rustiq	0	4	15	7,08
Ibm Eagle Q127	doustra	16	64	54	480,60
	lightSABRE_decay	0	4	16	12,69
	lightSABRE_lookahead	0	4	12	31,17
	pauliforest	-1	-1	-1	0,00
	qiskit_ai	0	4	16	90,62
	rustiq	0	3	13	11,49
Ibm Falcon Q27	doustra	20	79	62	225,47
	lightSABRE_decay	0	4	12	6,62
	lightSABRE_lookahead	0	3	19	22,67
	pauliforest	-1	-1	-1	0,00
	qiskit_ai	2	10	29	82,16
	rustiq	0	4	12	5,40
Ibm Heron Q133	doustra	16	64	54	496,98
	lightSABRE_decay	0	3	19	13,42
	lightSABRE_lookahead	0	2	24	34,03
	pauliforest	-1	-1	-1	0,00
	qiskit_ai	0	4	16	91,82
	rustiq	0	3	17	12,77
Ibm Manhattan Q65	doustra	16	64	54	309,99
	lightSABRE_decay	0	2	21	11,24
	lightSABRE_lookahead	0	4	12	24,75
	pauliforest	-1	-1	-1	0,00
	qiskit_ai	0	4	16	84,70
	rustiq	0	2	21	11,29

Continued on next page

Table 31 – continued from previous page

Topology	Optimizer	SWAP	CNOT	Depth	Time (ms)
Ibm Montreal Q27	doustra	42	163	84	225,73
	lightSABRE_decay	0	4	12	6,71
	lightSABRE_lookahead	0	2	23	26,06
	pauliforest	-1	-1	-1	0,00
	qiskit_ai	10	34	45	88,60
	rustiq	0	4	12	5,69
Ibm Paughkeepsie Q20	doustra	16	64	54	189,59
	lightSABRE_decay	0	2	20	9,89
	lightSABRE_lookahead	0	4	12	20,21
	pauliforest	-1	-1	-1	0,00
	qiskit_ai	0	4	16	81,85
	rustiq	0	2	20	9,11
Ibm Reuschlikon Q16	doustra	9	43	43	180,84
	lightSABRE_decay	0	2	21	11,27
	lightSABRE_lookahead	0	3	18	19,35
	pauliforest	-1	-1	-1	0,00
	qiskit_ai	0	4	16	81,78
	rustiq	0	2	22	9,90
Ibm Rochester Q53	doustra	16	64	54	275,80
	lightSABRE_decay	0	2	21	10,77
	lightSABRE_lookahead	0	3	17	24,65
	pauliforest	-1	-1	-1	0,00
	qiskit_ai	0	4	16	82,59
	rustiq	0	3	21	10,38
Ibm Tokyo Q20	doustra	13	64	43	276,08
	lightSABRE_decay	0	3	23	11,64
	lightSABRE_lookahead	0	2	24	25,07
	pauliforest	-1	-1	-1	0,00
	qiskit_ai	3	13	24	83,92
	rustiq	0	2	23	10,06
Ionq Harmony Q9	doustra	0	16	28	164,60
	lightSABRE_decay	0	4	12	49,69
	lightSABRE_lookahead	0	4	12	58,24
	pauliforest	-1	-1	-1	0,00
	qiskit_ai	0	4	16	85,11
	rustiq	0	4	12	47,96
Rigetti Novera Q9	doustra	16	67	47	178,10
	lightSABRE_decay	0	4	12	7,80
	lightSABRE_lookahead	0	3	22	22,69
	pauliforest	-1	-1	-1	0,00
	qiskit_ai	2	10	29	83,01
	rustiq	0	4	12	6,57

Continued on next page

Table 31 – continued from previous page

Topology	Optimizer	SWAP	CNOT	Depth	Time (ms)
Riken Fujitsu Q256	doustra	16	64	54	825,76
	lightSABRE_decay	0	3	17	49,27
	lightSABRE_lookahead	0	2	21	93,78
	pauliforest	-1	-1	-1	0,00
	qiskit_ai	0	4	16	120,78
	rustiq	0	3	17	48,43

Table 32. Full results for QAOA ansatz algorithm

Topology	Optimizer	SWAP	CNOT	Depth	Time (ms)
Google Willow Q105	doustra	22	94	64	95,65
	lightSABRE_decay	1	13	36	22,70
	lightSABRE_lookahead	1	13	36	42,68
	pauliforest	-1	-1	-1	0,00
	qiskit_ai	3	21	63	98,28
	rustiq	1	11	43	26,15
Hexagonal Lattice Q54	doustra	30	119	88	64,86
	lightSABRE_decay	2	12	46	21,09
	lightSABRE_lookahead	2	12	48	43,18
	pauliforest	-1	-1	-1	0,00
	qiskit_ai	6	30	81	91,61
	rustiq	2	14	43	16,55
Ibm Almaden Q20	doustra	17	74	69	70,58
	lightSABRE_decay	2	16	50	14,39
	lightSABRE_lookahead	2	14	49	35,27
	pauliforest	-1	-1	-1	0,00
	qiskit_ai	4	24	31	83,93
	rustiq	2	12	45	15,44
Ibm Cambridge Q28	doustra	34	123	73	72,22
	lightSABRE_decay	2	16	46	14,74
	lightSABRE_lookahead	2	14	45	35,94
	pauliforest	-1	-1	-1	0,00
	qiskit_ai	4	24	31	83,24
	rustiq	2	15	47	14,26
Ibm Eagle Q127	doustra	34	123	73	176,58
	lightSABRE_decay	2	16	41	16,91
	lightSABRE_lookahead	2	16	42	46,30
	pauliforest	-1	-1	-1	0,00
	qiskit_ai	4	24	31	94,57
	rustiq	2	16	39	16,44

Continued on next page

Table 32 – continued from previous page

Topology	Optimizer	SWAP	CNOT	Depth	Time (ms)
Ibm Falcon Q27	doustra	25	96	72	61,55
	lightSABRE_decay	2	14	59	17,84
	lightSABRE_lookahead	2	14	57	42,62
	pauliforest	-1	-1	-1	0,00
	qiskit_ai	3	21	65	84,34
	rustiq	2	14	41	13,94
Ibm Heron Q133	doustra	24	95	72	115,23
	lightSABRE_decay	1	13	46	34,93
	lightSABRE_lookahead	1	12	45	54,11
	pauliforest	-1	-1	-1	0,00
	qiskit_ai	4	24	31	100,42
	rustiq	2	16	40	17,57
Ibm Manhattan Q65	doustra	34	123	73	122,03
	lightSABRE_decay	2	16	36	13,68
	lightSABRE_lookahead	2	16	36	37,01
	pauliforest	-1	-1	-1	0,00
	qiskit_ai	4	24	31	90,23
	rustiq	2	16	41	13,58
Ibm Montreal Q27	doustra	51	193	98	143,44
	lightSABRE_decay	2	14	45	15,00
	lightSABRE_lookahead	2	14	45	35,42
	pauliforest	-1	-1	-1	0,00
	qiskit_ai	13	51	83	88,69
	rustiq	2	14	45	13,78
Ibm Paughkeepsie Q20	doustra	38	135	74	80,30
	lightSABRE_decay	2	14	38	14,59
	lightSABRE_lookahead	2	16	39	33,71
	pauliforest	-1	-1	-1	0,00
	qiskit_ai	4	24	31	82,20
	rustiq	2	14	40	12,68
Ibm Reuschlikon Q16	doustra	27	108	66	65,40
	lightSABRE_decay	1	11	39	13,27
	lightSABRE_lookahead	1	11	41	31,86
	pauliforest	-1	-1	-1	0,00
	qiskit_ai	4	24	54	84,16
	rustiq	1	11	47	13,26
Ibm Rochester Q53	doustra	34	123	73	112,14
	lightSABRE_decay	2	16	38	14,38
	lightSABRE_lookahead	2	18	37	37,82
	pauliforest	-1	-1	-1	0,00
	qiskit_ai	4	24	31	92,61
	rustiq	2	14	41	15,00

Continued on next page

Table 32 – continued from previous page

Topology	Optimizer	SWAP	CNOT	Depth	Time (ms)
Ibm Tokyo Q20	doustra	16	79	57	79,66
	lightSABRE_decay	0	6	45	16,61
	lightSABRE_lookahead	0	8	34	28,31
	pauliforest	-1	-1	-1	0,00
	qiskit_ai	6	30	57	86,03
	rustiq	0	6	45	16,28
Ionq Harmony Q9	doustra	0	22	28	46,86
	lightSABRE_decay	0	12	23	52,69
	lightSABRE_lookahead	0	12	23	61,30
	pauliforest	-1	-1	-1	0,00
	qiskit_ai	0	12	25	84,68
	rustiq	0	12	23	52,24
Rigetti Novera Q9	doustra	19	84	68	55,43
	lightSABRE_decay	1	11	43	18,72
	lightSABRE_lookahead	1	11	43	36,22
	pauliforest	-1	-1	-1	0,00
	qiskit_ai	2	18	50	85,32
	rustiq	1	13	37	14,33
Riken Fujitsu Q256	doustra	22	87	60	249,53
	lightSABRE_decay	1	13	45	44,52
	lightSABRE_lookahead	1	13	44	72,73
	pauliforest	-1	-1	-1	0,00
	qiskit_ai	4	24	31	126,30
	rustiq	1	11	43	47,48

Appendix 2

Used weights for validation on a real quantum computer

Table 33. Optimized variational parameters for the EfficientSU2 ansatz.

Weight	Value
$\theta[0]$	2,7533049717
$\theta[1]$	6,3735141614
$\theta[2]$	4,5992535801
$\theta[3]$	3,7614821919
$\theta[4]$	0,9802940293
$\theta[5]$	0,9801424782
$\theta[6]$	0,3649500986
$\theta[7]$	5,4423452326
$\theta[8]$	3,7769170097
$\theta[9]$	4,4489512172
$\theta[10]$	0,1293361921
$\theta[11]$	6,0941233324
$\theta[12]$	5,2303913697
$\theta[13]$	1,3341659804
$\theta[14]$	1,1424399624
$\theta[15]$	1,1523645216

Appendix 3
Generalisation rules results tables

Table 34. Evaluation of the generalisation rules performance for CNOT gates count optimisation

Topology	Circuit	Expected Mapper	Selected Mapper	Correct
IBM Reuschlikon Q16	Demo	Lookahead	rustiq	false
IBM Tokyo Q20	Demo	Decay	rustiq	false
IBM Paughkeepsie Q20	Demo	Lookahead	lightSABRE lookahead	true
IBM Cambridge Q28	Demo	Rustiq/Lookahead	lightSABRE lookahead	true
IBM Montreal Q27	Demo	Lookahead/decay/rustiq	lightSABRE lookahead	true
IBM Almaden Q20	Demo	Lookahead	lightSABRE lookahead	true
IBM Rochester Q53	Demo	Lookahead	lightSABRE lookahead	true
IBM Manhattan Q65	Demo	Rustiq	lightSABRE lookahead	false
IBM Heron Q133	Demo	Rustiq	lightSABRE lookahead	false
IBM Eagle Q127	Demo	Rustiq/Lookahead	lightSABRE lookahead	true
IBM Falcon Q27	Demo	Decay/Lookahead	lightSABRE lookahead	true
Rigetti Novera Q9	Demo	PauliForest	rustiq	false
IonQ Harmony Q9	Demo	PauliForest	pauliforest	true
Google Willow Q105	Demo	Decay	rustiq	false
Hexagonal Lattice Q54	Demo	Lookahead	rustiq	false
Riken Fujitsu Q256	Demo	Lookahead	rustiq	false
IBM Reuschlikon Q16	LiH	Decay	rustiq	false
IBM Tokyo Q20	LiH	Rustiq	rustiq	true
IBM Paughkeepsie Q20	LiH	Lookahead	rustiq	false
IBM Cambridge Q28	LiH	Decay	rustiq	false
IBM Montreal Q27	LiH	Lookahead	rustiq	false
IBM Almaden Q20	LiH	Rustiq	rustiq	true
IBM Rochester Q53	LiH	Rustiq	rustiq	true
IBM Manhattan Q65	LiH	Rustiq	rustiq	true
IBM Heron Q133	LiH	Decay	rustiq	false
IBM Eagle Q127	LiH	Rustiq	rustiq	true
IBM Falcon Q27	LiH	Rustiq	rustiq	true
Google Willow Q105	LiH	Lookahead	rustiq	false
Hexagonal Lattice Q54	LiH	Lookahead	rustiq	false
Riken Fujitsu Q256	LiH	Rustiq	rustiq	true
IBM Reuschlikon Q16	H2	PauliForest	rustiq	false

Table 34. Evaluation of the generalisation rules performance for CNOT gates count optimisation

Topology	Circuit	Expected Mapper	Selected Mapper	Correct
IBM Tokyo Q20	H2	PauliForest	rustiq	false
IBM Paughkeepsie Q20	H2	PauliForest	lightSABRE lookahead	false
IBM Cambridge Q28	H2	PauliForest	lightSABRE lookahead	false
IBM Montreal Q27	H2	Decay	lightSABRE lookahead	false
IBM Almaden Q20	H2	PauliForest	lightSABRE lookahead	false
IBM Rochester Q53	H2	Decay/Lookahead	lightSABRE lookahead	true
IBM Manhattan Q65	H2	Decay	lightSABRE lookahead	false
IBM Heron Q133	H2	Lookahead	lightSABRE lookahead	true
IBM Eagle Q127	H2	PauliForest	lightSABRE lookahead	false
IBM Falcon Q27	H2	PauliForest	lightSABRE lookahead	false
Rigetti Novera Q9	H2	Decay	rustiq	false
IonQ Harmony Q9	H2	PauliForest	pauliforest	true
Google Willow Q105	H2	Lookahead	rustiq	false
Hexagonal Lattice Q54	H2	Decay	rustiq	false
Riken Fujitsu Q256	H2	Rustiq	rustiq	true
IBM Reuschlikon Q16	QAOA Hamil	Lookahead	rustiq	false
IBM Tokyo Q20	QAOA Hamil	Rustiq	rustiq	true
IBM Paughkeepsie Q20	QAOA Hamil	Decay/Lookahead	lightSABRE lookahead	true
IBM Cambridge Q28	QAOA Hamil	Decay/Lookahead	lightSABRE lookahead	true
IBM Montreal Q27	QAOA Hamil	Decay/Rustiq	lightSABRE lookahead	false
IBM Almaden Q20	QAOA Hamil	Decay/Lookahead	lightSABRE lookahead	true
IBM Rochester Q53	QAOA Hamil	Rustiq/Lookahead-/Decay	lightSABRE lookahead	true
IBM Manhattan Q65	QAOA Hamil	Decay	lightSABRE lookahead	false
IBM Heron Q133	QAOA Hamil	Rustiq	lightSABRE lookahead	false
IBM Eagle Q127	QAOA Hamil	Decay/Lookahead	lightSABRE lookahead	true
IBM Falcon Q27	QAOA Hamil	Rustiq	lightSABRE lookahead	false

Table 34. Evaluation of the generalisation rules performance for CNOT gates count optimisation

Topology	Circuit	Expected Mapper	Selected Mapper	Correct
Rigetti Novera Q9	QAOA Hamil	Rustiq/Lookahead-/Decay	rustiq	true
IonQ Harmony Q9	QAOA Hamil	Rustiq/Lookahead-/Decay/AI	pauliforest	false
Google Willow Q105	QAOA Hamil	Rustiq	rustiq	true
Hexagonal Lattice Q54	QAOA Hamil	Decay	rustiq	false
Riken Fujitsu Q256	QAOA Hamil	Rustiq	rustiq	true
IBM Reuschlikon Q16	QAOA Ansatz	Decay/lookahead	lightSABRE lookahead	true
IBM Tokyo Q20	QAOA Ansatz	Decay/Rustiq	lightSABRE lookahead	false
IBM Paughkeepsie Q20	QAOA Ansatz	Decay/Rustiq (prastesnis depth)	lightSABRE lookahead	false
IBM Cambridge Q28	QAOA Ansatz	Lookahead	lightSABRE lookahead	true
IBM Montreal Q27	QAOA Ansatz	Rustiq/Lookahead-/Decay	lightSABRE lookahead	true
IBM Almaden Q20	QAOA Ansatz	Rustiq	lightSABRE lookahead	false
IBM Rochester Q53	QAOA Ansatz	Decay (rustiq, lookahead)	lightSABRE lookahead	true
IBM Manhattan Q65	QAOA Ansatz	Decay/Lookahead	lightSABRE lookahead	true
IBM Heron Q133	QAOA Ansatz	Lookahead	lightSABRE lookahead	true
IBM Eagle Q127	QAOA Ansatz	Rustiq	lightSABRE lookahead	false
IBM Falcon Q27	QAOA Ansatz	Rustiq	lightSABRE lookahead	false
Rigetti Novera Q9	QAOA Ansatz	Rustiq/Lookahead-/Decay/	lightSABRE lookahead	true
IonQ Harmony Q9	QAOA Ansatz	Rustiq	lightSABRE lookahead	false
Google Willow Q105	QAOA Ansatz	Rustiq	lightSABRE lookahead	false
Hexagonal Lattice Q54	QAOA Ansatz	Decay/Lookahead	lightSABRE lookahead	true
Riken Fujitsu Q256	QAOA Ansatz	Rustiq	lightSABRE lookahead	false
IBM Reuschlikon Q16	Efficient Su2	Decay/Rustiq	lightSABRE lookahead	false
IBM Tokyo Q20	Efficient Su2	Decay	lightSABRE lookahead	false

Table 34. Evaluation of the generalisation rules performance for CNOT gates count optimisation

Topology	Circuit	Expected Mapper	Selected Mapper	Correct
IBM Paughkeepsie Q20	Efficient Su2	Decay/Rustiq	lightSABRE lookahead	false
IBM Cambridge Q28	Efficient Su2	Decay/Lookahead	lightSABRE lookahead	true
IBM Montreal Q27	Efficient Su2	Lookahead	lightSABRE lookahead	true
IBM Almaden Q20	Efficient Su2	Decay/Rustiq	lightSABRE lookahead	false
IBM Rochester Q53	Efficient Su2	Decay	lightSABRE lookahead	false
IBM Manhattan Q65	Efficient Su2	Decay/Rustiq	lightSABRE lookahead	false
IBM Heron Q133	Efficient Su2	Lookahead	lightSABRE lookahead	true
IBM Eagle Q127	Efficient Su2	Rustiq	lightSABRE lookahead	false
IBM Falcon Q27	Efficient Su2	Rustiq	lightSABRE lookahead	false
Rigetti Novera Q9	Efficient Su2	Rustiq/Lookahead-/Decay/AI	lightSABRE lookahead	true
IonQ Harmony Q9	Efficient Su2	Lookahead/Rustiq	lightSABRE lookahead	true
Google Willow Q105	Efficient Su2	Decay/Rusitq	lightSABRE lookahead	false
Hexagonal Lattice Q54	Efficient Su2	Lookahead	lightSABRE lookahead	true
Riken Fujitsu Q256	Efficient Su2	Lookahead	lightSABRE lookahead	true

Table 35. Evaluation of the generalisation rules performance for circuit depth optimisation

Topology	Circuit	Expected Mapper	Selected Mapper	Correct
IBM Reuschlikon Q16	Demo	PauliForest	PauliForest	true
IBM Tokyo Q20	Demo	PauliForest	PauliForest	true
IBM Paughkeepsie Q20	Demo	PauliForest	PauliForest	true
IBM Cambridge Q28	Demo	PauliForest	PauliForest	true
IBM Montreal Q27	Demo	PauliForest	PauliForest	true
IBM Almaden Q20	Demo	PauliForest	PauliForest	true
IBM Rochester Q53	Demo	PauliForest	PauliForest	true
IBM Manhattan Q65	Demo	PauliForest	PauliForest	true
IBM Heron Q133	Demo	PauliForest	PauliForest	true
IBM Eagle Q127	Demo	PauliForest	PauliForest	true

Topology	Circuit	Expected Mapper	Selected Mapper	Correct
IBM Falcon Q27	Demo	PauliForest	PauliForest	true
Rigetti Novera Q9	Demo	PauliForest	PauliForest	true
IonQ Harmony Q9	Demo	PauliForest	PauliForest	true
Google Willow Q105	Demo	PauliForest	PauliForest	true
Hexagonal Lattice Q54	Demo	PauliForest	PauliForest	true
Riken Fujitsu Q256	Demo	PauliForest	PauliForest	true
IBM Reuschlikon Q16	LiH	Decay	PauliForest	false
IBM Tokyo Q20	LiH	Lookahead	PauliForest	false
IBM Paughkeepsie Q20	LiH	PauliForest	PauliForest	true
IBM Cambridge Q28	LiH	PauliForest	PauliForest	true
IBM Montreal Q27	LiH	Lookahead	PauliForest	false
IBM Almaden Q20	LiH	PauliForest	PauliForest	true
IBM Rochester Q53	LiH	PauliForest	PauliForest	true
IBM Manhattan Q65	LiH	PauliForest	PauliForest	true
IBM Heron Q133	LiH	PauliForest	PauliForest	true
IBM Eagle Q127	LiH	PauliForest	PauliForest	true
IBM Falcon Q27	LiH	PauliForest	PauliForest	true
Google Willow Q105	LiH	Lookahead	PauliForest	false
Hexagonal Lattice Q54	LiH	PauliForest	PauliForest	true
Riken Fujitsu Q256	LiH	Rustiq	PauliForest	false
IBM Reuschlikon Q16	H2	PauliForest	PauliForest	true
IBM Tokyo Q20	H2	PauliForest	PauliForest	true
IBM Paughkeepsie Q20	H2	PauliForest	PauliForest	true
IBM Cambridge Q28	H2	PauliForest	PauliForest	true
IBM Montreal Q27	H2	PauliForest	PauliForest	true
IBM Almaden Q20	H2	PauliForest	PauliForest	true
IBM Rochester Q53	H2	PauliForest	PauliForest	true
IBM Manhattan Q65	H2	PauliForest	PauliForest	true
IBM Heron Q133	H2	PauliForest	PauliForest	true
IBM Eagle Q127	H2	PauliForest	PauliForest	true
IBM Falcon Q27	H2	PauliForest	PauliForest	true
Rigetti Novera Q9	H2	PauliForest	PauliForest	true
IonQ Harmony Q9	H2	PauliForest	PauliForest	true
Google Willow Q105	H2	PauliForest	PauliForest	true
Hexagonal Lattice Q54	H2	PauliForest	PauliForest	true
Riken Fujitsu Q256	H2	PauliForest	PauliForest	true
IBM Reuschlikon Q16	QAOA	AI	AI	true
IBM Tokyo Q20	QAOA	AI	AI	true
IBM Paughkeepsie Q20	QAOA	PauliForest	AI	false
IBM Cambridge Q28	QAOA	AI	AI	true
IBM Montreal Q27	QAOA	Decay	AI	false
IBM Almaden Q20	QAOA	AI	AI	true
IBM Rochester Q53	QAOA	AI	AI	true
IBM Manhattan Q65	QAOA	AI	AI	true
IBM Heron Q133	QAOA	AI	AI	true
IBM Eagle Q127	QAOA	AI	AI	true

Topology	Circuit	Expected Mapper	Selected Mapper	Correct
IBM Falcon Q27	QAOA	Lookahead	AI	false
Rigetti Novera Q9	QAOA	Rustiq/Lookahead-/Decay/AI	AI	true
IonQ Harmony Q9	QAOA	PauliForest/Lookahead	AI	false
Google Willow Q105	QAOA	PauliForest	AI	false
Hexagonal Lattice Q54	QAOA	Decay/Rustiq	AI	false
Riken Fujitsu Q256	QAOA	AI	AI	true
IBM Reuschlikon Q16	QAOA Ansatz	Decay	Rustiq	false
IBM Tokyo Q20	QAOA Ansatz	Lookahead	Rustiq	false
IBM Paughkeepsie Q20	QAOA Ansatz	AI	Rustiq	false
IBM Cambridge Q28	QAOA Ansatz	AI	Rustiq	false
IBM Montreal Q27	QAOA Ansatz	Decay/Rustiq	Rustiq	true
IBM Almaden Q20	QAOA Ansatz	AI	Rustiq	false
IBM Rochester Q53	QAOA Ansatz	AI	Rustiq	false
IBM Manhattan Q65	QAOA Ansatz	AI	Rustiq	false
IBM Heron Q133	QAOA Ansatz	AI	AI	true
IBM Eagle Q127	QAOA Ansatz	AI	AI	true
IBM Falcon Q27	QAOA Ansatz	Rustiq	Rustiq	true
Rigetti Novera Q9	QAOA Ansatz	Rustiq	Rustiq	true
IonQ Harmony Q9	QAOA Ansatz	Rustiq/Lookahead-/Decay	Rustiq	true
Google Willow Q105	QAOA Ansatz	Decay/Lookahead	AI	false
Hexagonal Lattice Q54	QAOA Ansatz	Rustiq	Rustiq	true
Riken Fujitsu Q256	QAOA Ansatz	AI	AI	true
IBM Reuschlikon Q16	Efficient SU2	Lookahead	Rustiq	false
IBM Tokyo Q20	Efficient SU2	Decay/Rustiq	Rustiq	true
IBM Paughkeepsie Q20	Efficient SU2	Lookahead	Rustiq	false
IBM Cambridge Q28	Efficient SU2	Decay/Rustiq	Rustiq	true
IBM Montreal Q27	Efficient SU2	Decay/Rustiq	Rustiq	true
IBM Almaden Q20	Efficient SU2	Lookahead	Rustiq	false
IBM Rochester Q53	Efficient SU2	AI	Rustiq	false
IBM Manhattan Q65	Efficient SU2	Decay/Rustiq	Rustiq	true
IBM Heron Q133	Efficient SU2	Rustiq	AI	false
IBM Eagle Q127	Efficient SU2	Lookahead	AI	false
IBM Falcon Q27	Efficient SU2	Decay/Rustiq	Rustiq	true
Rigetti Novera Q9	Efficient SU2	Decay/Rustiq	Rustiq	true
IonQ Harmony Q9	Efficient SU2	Rustiq/Lookahead-/Decay	Rustiq	true
Google Willow Q105	Efficient SU2	Lookahead	AI	false
Hexagonal Lattice Q54	Efficient SU2	Rustiq	Rustiq	true
Riken Fujitsu Q256	Efficient SU2	AI	AI	true