POLITECNICO DI TORINO

Master degree course in Quantum Engineering

Master Degree Thesis

PANZEROTTO

Probabilistic trANspiler and optimiZER for OpTimal statevecTOrs

Supervisors
Prof. Giovanna Turvani
Ph.D. Deborah Volpe
Prof. Mariagrazia Graziano

Candidate
Pietro Antezza

March 2026

Summary

Noisy Intermediate-Scale Quantum (NISQ) computers are potentially capable of solving Non-deterministic Polynomial-time hard problems that remain computationally intractable for classical computers. However, the potential of NISQ computers cannot be fully exploited due to physical limitations: state-of-the-art quantum hardware backends suffer from high error rates and short decoherence times. As a consequence, the probability of a successful computation is inversely proportional to the depth and gate count of the quantum circuit; such metrics, when sufficiently high, do not allow the system to complete the computation within the coherence time. Compilation and Optimization techniques become necessary to overcome these physical constraints and to achieve a better success probability. Recent research has shifted toward software-based solutions for the entire quantum stack, focusing on pre-execution circuit optimization. Traditional approaches, such as peephole optimization, template-based compilers, and rule-based transpilers, rely on predefined sequences of gate substitutions. Although effective at a local level, these methods often fail to capture the global structure of the circuit, missing significant optimization paths. Moreover, global synthesis of circuits is often based on brute-force approaches, requiring high computational resources and time to explore the large search space of equivalent circuits. The aim of this thesis is to propose a new approach for quantum circuit optimization by exploring state-of-the-art Generative Artificial Intelligence (GenAI) techniques. Specifically, this thesis introduces PANZEROTTO, a framework that addresses circuit optimization as a generative task rather than a transformation one. Through the use of a Directed Acyclic Graph Variational Autoencoder (DAG-VAE), the compiler learns both the global and local topological structures of input circuits, generating new, optimized versions. The main objective of this model is to reduce both gate count and circuit depth, accepting a trade-off in terms of circuit logical fidelity, to maximize the utility of short-lived NISQ systems. This research explores a possible alternative methodology designed to reduce computational complexity by adopting a generalized approach: the model is trained on low-dimensional quantum circuits but with the capabilities to be used for high-dimensional ones. Exploiting the stochastic exploration of GenAI, the framework can sample multiple candidate circuits to find the optimal configuration. Furthermore, the project aims to unify

hardware-aware and hardware-unaware optimization through the integration of Reinforcement Learning and reward shaping techniques. The model has been tested on different datasets, reporting the same results every time: the Loss Function never drops below a certain threshold, highlighting the limitations of the architecture to learn and generate quantum circuits. The best results reported an average of 80% reconstruction fidelity with an associated maximum of 5% reduction in circuit depth. This thesis reports some important features from the architectural point of view such as a Hybrid Training technique and the Wire Pooling mechanism; on the other hand, the model lacks the capabilities needed to tackle such a complex problem as quantum circuit optimization. Further steps in this project will be to enrich the quality of information given to the model to enhance its performance, or to test a full training procedure with Reinforcement Learning.

Contents

Chapter 1

Introduction

The transition from theoretical algorithms to quantum circuit execution on real quantum hardware backends is directly affected by the physical limitations of qubits; state of the art quantum hardware suffers from high error rates and short decoherence times[1]. It follows that the probability of a successful computation is inversely proportional to the depth and gate count of the quantum circuit. The higher the gate count and depth, the higher the probability of decoherence occurring.

Quantum circuit compilation and optimization address this problem with a vast variety of techniques, from brute force approaches to state-of-the-art machine learning techniques. Traditional peephole optimization (or template based compilers) and rule based transpiler rely on predefined sequences; while locally efficient, these methods fail to capture the global structure of the circuit, missing possible optimization paths. On the other hand, exact global synthesis methods rely on brute force approaches, resulting in computational intractability for large circuits. Machine Learning and Deep Learning techniques are powerful alternatives designed for specific use cases exploiting different data representations for efficient exploration in the space of equivalent circuits. After a quick review of basic concepts for Quantum Computing, state-of-the-art techniques for both strategies are reviewed in Part I.

The main reason for this research is to propose an alternative approach for quantum circuit optimization to *reduce the computational complexity*, proposing a generalized approach trained on low dimensional circuits, but still valid on high-dimensional ones; to leverage *stochastic exploration* of Generative-AI, sampling multiple candidates; to *unify hardware-aware and hardware-unaware optimization* by leveraging Reinforcement Learning and reward shaping techniques.

PANZEROTTO addresses the problems of NISQ devices with a fundamentally distinct paradigm, as a generative task. By exploiting a Directed Acyclic Graph Variational Autoencoder (DAG-VAE), the compiler learns the global and local

structure of a circuit, generating a new optimized version of the circuit. Furthermore, this project proposes also a full structure for a compiler; including data preprocessing, circuit validation through different simulators and data post-processing. The entire compiler architecture and model design choices are detailed in Part II.

In Part III, the results for different initial datesets of circuits are reported, obtained by Principal Component Analysis (PCA) and studying the behavior of the compenents of the Loss Function with respect to the epochs in the training process.

Lastly, the final conclusions of the work and possible future improvements and research paths are discussed in Part IV.

Part I

Preliminary knowledge

Chapter 2

Quantum Computing Basis

In this chapter basis concepts and terminology are introduced. The following review is not meant to be satisfactory from a theoretical point of view, the only scope is to define what later is assumed to be trivial.

For a full review of Quantum Information Theory, see [2].

2.1 Qubit

In Quantum Computing the basic computational unit is the *qubit*, it is the equivalent of the bit in digital electronics, but with a completely different technology behind. While a bit can be only in the state *0* or *1*, a qubit can also exist in any superposition of the two classical states.

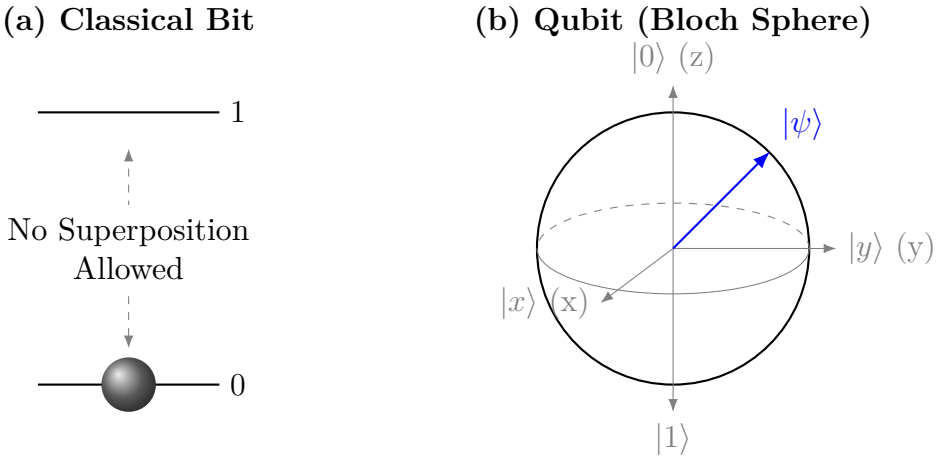


Figure 2.1: Comparison between a classical bit (a) and a qubit (b). Classical bit can exist only in the discrete states 0 or 1. The qubit can be represented as a vector on the Bloch sphere, allowing superposition.

2.2 Quantum Gates

Quantum Gates are the operations applicable to qubits, and they can be considered as the quantum equivalent of the classical digital gates. From a mathematical point of view, a qubit is represented as a unitary vector in a bi-dimensional Hilbert space, while quantum gates are described by unitary matrices. It follows that applying a quantum gate to a qubit means multiplying by left a matrix to a vector, obtaining a new vector representing the qubit after the application of the gate. Some quantum gates and their mathematical representation are reported in Equation 2.1, Equation 2.2, Equation 2.3 and Equation 2.4.

$$H \quad \equiv \quad |q\rangle \text{ —}\boxed{H}\text{— } \quad \equiv \quad \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix} \tag{2.1}$$

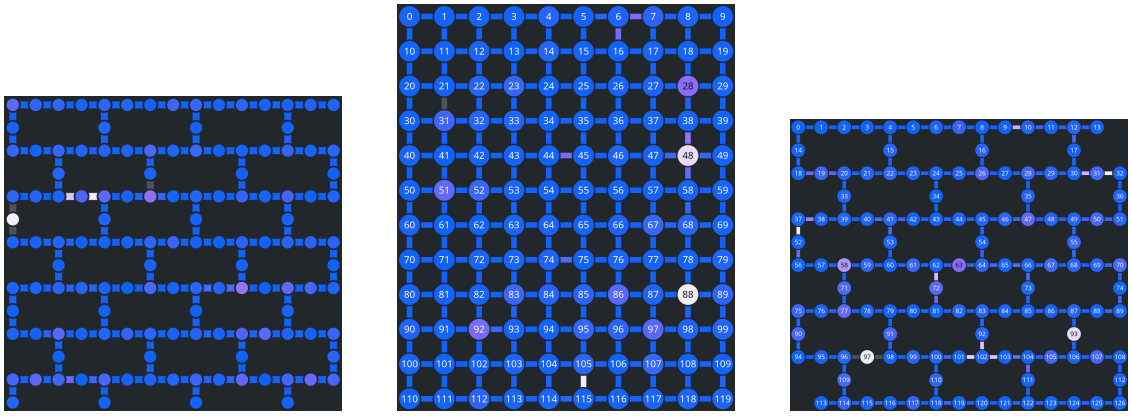
$$CNOT \quad \equiv \quad \begin{array}{l}|c\rangle\\ |t\rangle\end{array} \quad \equiv \quad \begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{pmatrix} \tag{2.2}$$

$$SWAP \quad \equiv \quad \begin{array}{l}|q_0\rangle\\ |q_1\rangle\end{array} \quad \equiv \quad \begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\end{pmatrix} \tag{2.3}$$

$$CCNOT \quad \equiv \quad \begin{array}{l}|q_0\rangle\\ |q_1\rangle\\ |q_2\rangle\end{array} \quad \equiv \quad \begin{pmatrix}1 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1\\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0\end{pmatrix} \tag{2.4}$$

2.3 Quantum Processing Unit (QPU)

A *Quantum Processing Unit (QPU)* is an ensemble of qubits organized in different ways. The main parameter to take into account when talking about QPUs is the *Coupling Map*, which represents each connection among the qubits with associated characteristics; some real examples are reported in Figure 2.2.



(a) ibm_torino (b) ibm_miami (c) ibm_brussels

Figure 2.2: Coupling maps for three different QPUs designed by IBM. Each dot represents a qubit, each line is a connection between qubits. The color gradient in both components is a visual representation for quality of qubits/connections. [3]

2.4 Quantum Circuits

Quantum Circuits are defined as sequences of quantum gates. Are used as intermediate representation between an abstract algorithm and hardware level instructions for QPU. Some examples are reported in Figure 2.3. The main parameters for a quantum circuit are the following:

- Number of qubits
- Gate Counts: total number of quantum gates in the circuit.
- Depth: number of layers of quantum gates required to complete the computation.
- Fidelity: Logical or unitary fidelity, a measure of the distance between the unitary representation of two different circuits.

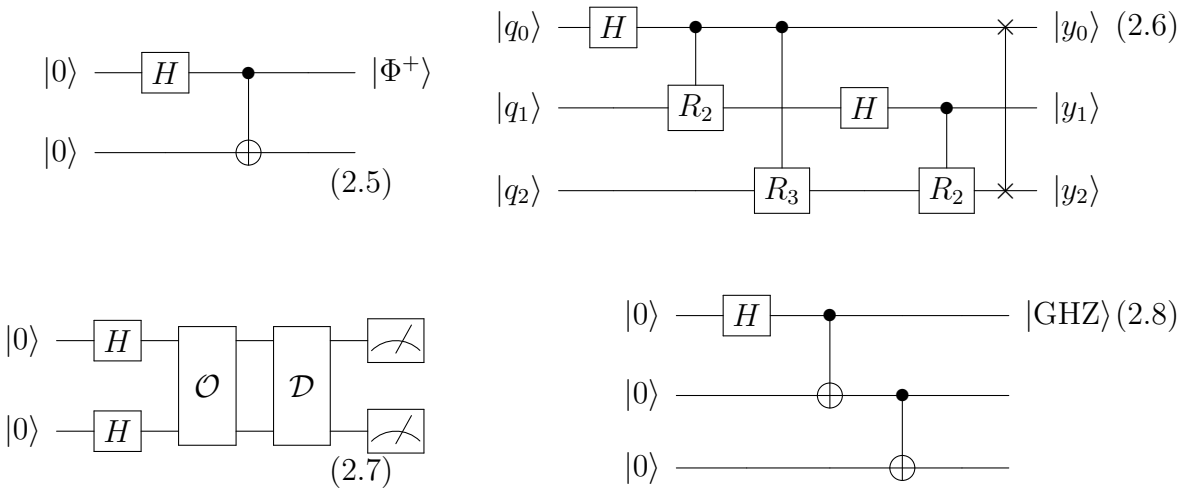


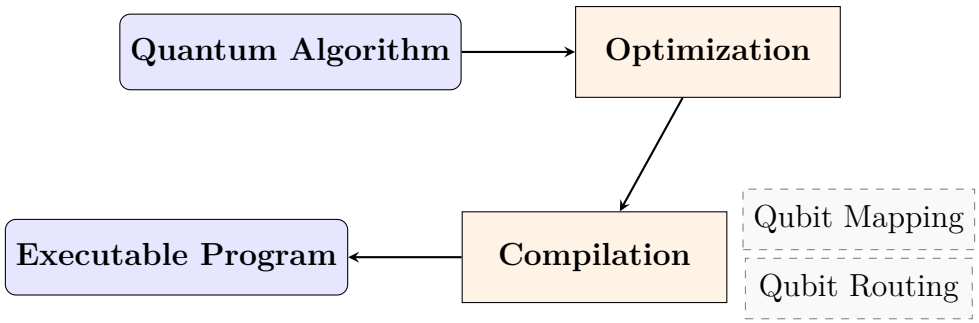
Figure 2.3: Quantum circuits examples represented in the gate model. Equation 2.5 is the initialization of a Bell state, Equation 2.6 is a simple Quantum Fourier Transform (QFT), Equation 2.7 is an example for Grover algorithm where $\mathcal{O}$ is the Oracle and $\mathcal{D}$ the diffuser,Equation 2.8 is the GHZ state preparation.

Chapter 3

Quantum Circuits optimization

3.1 State of the art for quantum circuit compilation and optimization

Starting from a logic circuit designed based on a quantum algorithm, two main steps are required in order to obtain an executable program on quantum hardware, namely optimization and compilation; the former is not mandatory, but extremely useful to target state-of-the-art quantum hardware; the latter, usually divided into qubit mapping and qubit routing, is, on the other hand, necessary to fit the quantum circuit to the hardware backend constraints[4].



3.1.1 Quantum Circuit Representation

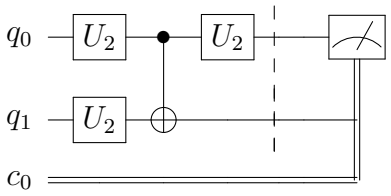
The most common method to represent a quantum circuit is known as the Gate Model, namely describing the algorithm as a sequence of quantum gates with their own parameters applied on specific qubits (3.1). The Gate Model is universally recognized as the standard quantum circuit description language, thanks to the fact that it is human-friendly and easy to manage; nonetheless it has some drawbacks

when one wants to optimize the algorithm. In fact, there are multiple representations that enlighten and simplify algorithms that in the Gate Model would have been much more difficult. The main ones are:

- Direct Acyclic Graph (DAG), which treats each quantum gate as a node in the graph exploiting directed edges to capture complex dependencies between nodes in the graph[5];
- phase polynomials, a mathematical method focused on the phase of the quantum state[6];
- tensor networks[7];
- ZX diagrams[8].

OpenQASM 2.0

```
OPENQASM 2.0;
include "qelib1.inc";
qreg q[2];
creg c[1];
u2(0,0) q[0];
u2(-pi,-pi) q[1];
cx q[0],q[1];
u2(-pi,-pi) q[0];
barrier q[0],q[1];
measure q[0] -> c[0];
```



(3.1)

Figure 3.1: OpenQASM code and correspondent visual representation.

3.1.2 Priorities in circuit optimization

Quantum circuit optimization should be designed by carefully choosing the best technique to tackle the specific problem that affects the system considered. In the Noisy Intermediate-Scale Quantum (NISQ) era the biggest problem is qubit decoherence (a process where the environment interacts with a qubit, causing it to lose its local quantum information, collapsing the wavefunction[2] as in Figure 3.2). Therefore, the main parameters to be taken into account are circuit depth, gate count and CNOT count; in Fault Tolerant Quantum Computing (FTQC) era the problem will be reduced to optimize the number of T gate, which is the one that requires the biggest effort in error correction codes.

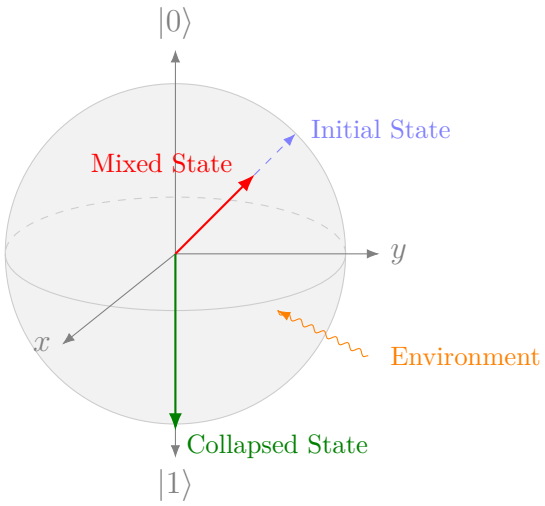


Figure 3.2: Decoherence effect represented on the Bloch Sphere: environment action reduces Bloch vector length (red arrow), or it collapses the vector to one of the basis states (green arrow).

3.1.3 Optimization levels

Hardware Unaware

The optimization of a quantum circuit begins at the mathematical level; optimizing the sequence of quantum gates is essential to minimize both resource usage and error propagation. Most works in this context exploit techniques based on pattern matching and substitution, ZX-calculus, or decision diagrams.

In [9], the authors developed a classical algorithm for pattern matching that efficiently finds all possible substitutions for circuit sizes suitable for near-term quantum backends. By identifying mathematical equivalences between different gates, as in Figure 3.3, a compiler can replace complex subcircuits with simpler and shorter alternatives. Within this framework, simplification rules exploit the algebraic properties of quantum operations, such as unitarity (3.2) or commutativity (3.3) [2]. These properties allow, for example, gate cancellation, i.e., eliminating subsequent gates that result in an identity matrix, or merging of rotation gates [10].

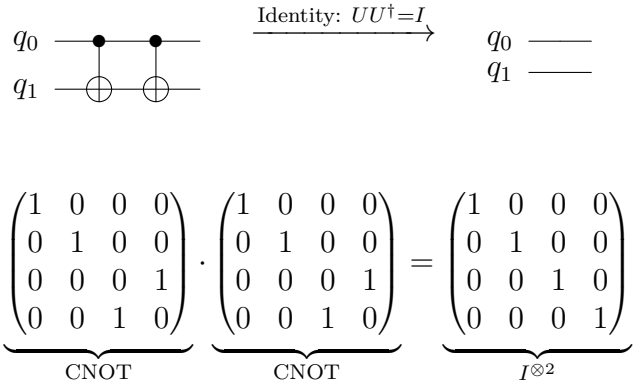
$$UU^{\dagger} = I \tag{3.2}$$

$$[A, B] = AB - BA = 0 \tag{3.3}$$

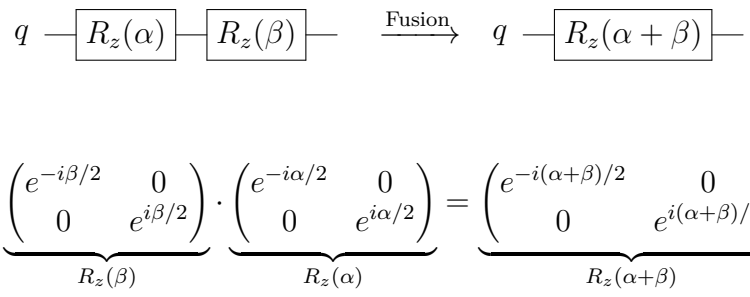
Beyond gate-level operations, [11] introduced PyZX, a tool that converts the circuit into a ZX-graph, then optimizes using graph theory rules to remove nodes, allowing significant compression of both T-count and CNOT-count.

The approach followed by [12] is based on Decision Diagrams, exploiting structural redundancies in quantum states. Although primarily designed for efficient

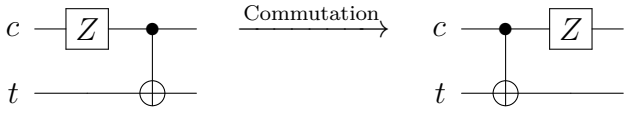
simulation on classical hardware, this approach also plays an important role in hardware-agnostic optimization by compacting the representation of quantum operations, making it useful for minimizing circuits for near-term applications.



(a) **Gate Cancellation**: Two consecutive CNOT (or any quantum gate) nullify each other.



(b) **Gate Fusion**: Consecutive rotation on the same axis can be unified summing angles.

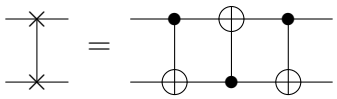


(c) **Commutation**: Some quantum gates commute with each other, e.g. $[Z \otimes I, \text{CNOT}] = 0$; this allows for future cancellation or merging.

Figure 3.3: Examples for pattern matching quantum circuit optimization.

Hardware Aware

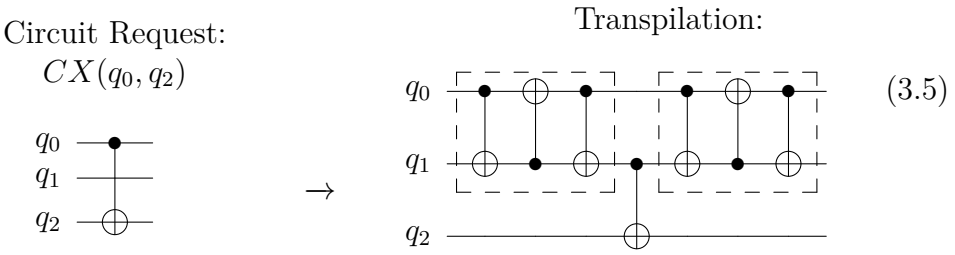
Hardware aware optimization level is the stage where logical quantum circuit are adapted to the hardware backend according to its physical constraints. The primary challenge at this level is fitting the used qubit at the logical level, in which we assume an ideal machine with all qubits interacting without constraints, to physical qubits that in most of the cases are not fully connected, but limited, depending on the configuration and the technology utilized. Indeed, not every qubit is connected with each other, resulting in a NP-hard problem. Usually an initial mapping is performed with a one-to-one correspondence between logical and physical qubits, then solving connectivity issues inserting in the circuits SWAP gates to apply the quantum gates as designed. SWAP gates are a powerful tool at this stage but, being composed of three CNOT gates as in (3.4), add a very high computational overhead significantly increasing circuit depth and gate count. It might be helpful to think SWAP gates as real time operations repositioning qubits at the logical level to satisfy the constraints of the target operation, resulting in the exchange of the statevector of the two qubits with no hardware modifications required.



(3.4)

Moreover, physical phenomena must be taken in consideration at the hardware aware level, namely decoherence, that affects inevitably every physical qubit, and gate error that can be dependent on the specific qubit in a particular QPU, enabling the compiler to choose which qubit is more suitable to perform the quantum gate taken in consideration.

State of the art compilers use heuristic search (e.g. Dijkstra) or Reinforcement Learning (RL) agents to create an optimal SWAP strategy.



(3.5)

Figure 3.4: Hardware Aware Traspilation Example: to execute a CNOT between q_0 e q_2 (not connected), the compiler insert SWAPs to bring the logical state q_0 to a qubit physically connected to q_2. SWAP usage to solve connectivity issues increases significantly the circuit depth.

3.2 Qiskit

Proposed by IBM in 2017, Qiskit is a comprehensive toolchain for the Quantum Computing software stack. It embeds the representation of abstract quantum circuits, their synthesis into concrete gates, optimization and transformation into Instruction Set Architecture (ISA) circuits compatible with specific quantum hardware. The central data structure in Qiskit is the quantum circuit, it is designed in order to manage various levels of abstraction, from mathematical operators to physical pulse-level operations. The whole architecture is depicted in Figure 3.5. Since its launch, Qiskit has been widely adopted for educational, research and industrial application, thanks to its modularity and extensibility. In fact, while the user interacts with a list of instruction, Qiskit internally represents circuits using DAGs to facilitate analysis and rewriting. To optimize and compile quantum circuits a transpiler is used; it is an optimized circuit-to-circuit compiler with extensive configuration options, it can be used just as an optimizer or just as a compiler, based on the target devices that can be a real QPU or an abstract backend with the same gate set of the input circuit. Moreover, Qiskit could perform circuit synthesis, routing and layout optimization, gate optimization based on substitution and commutation.[13]

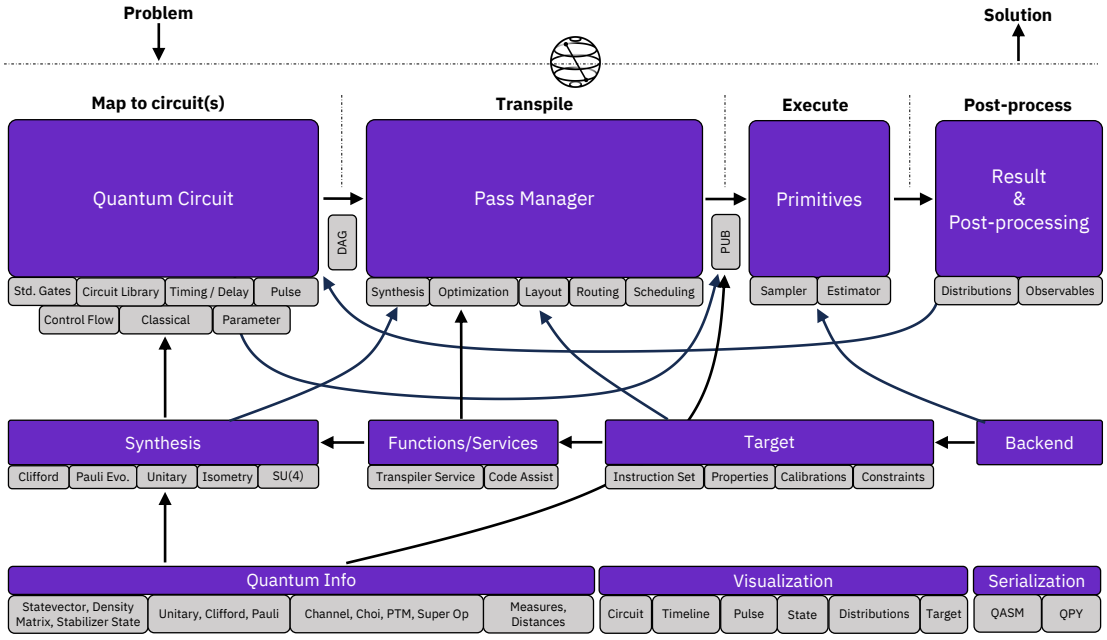


Figure 3.5: Qiskit's software architecture. The core component in Qiskit is the quantum circuit, around which the rest of the framework revolves. The transpiler transforms quantum circuits by applying a pipeline of passes on them, orchestrated by a pass manager. Circuit transformations occur according to a target, which is an abstract machine model that summarizes the pertinent features of a backend[13].

3.3 t|ket>

Developed originally by Cambridge Quantum Computing (Quantinuum), t|ket>[14] is a quantum circuit compiler and optimizer designed as a retargetable toolchain, able to translate complex algorithms in various languages for a wide range of QPUs. t|ket>leverages advanced techniques from graph theory and ZX-calculus, allowing simplification and optimization of quantum circuits through non-intuitive rules at the gate level. The compiler performs a first step in which local and global optimization are applied to reduce the number of two-qubits gates and T gates; followed by a routing phase that adapts the circuit to the target hardware, minimizing SWAP gate count.

3.4 Manfredi master thesis

Previously, at Politecnico di Torino a work in this sector was initiated in [15], a three-stage compilation toolchain.

The framework begins with template based QASM optimization, followed by technology dependent gate compaction(leveraging hardware specific substitution or commutation rules), and finally two-qubit gate decomposition according to the target technology.

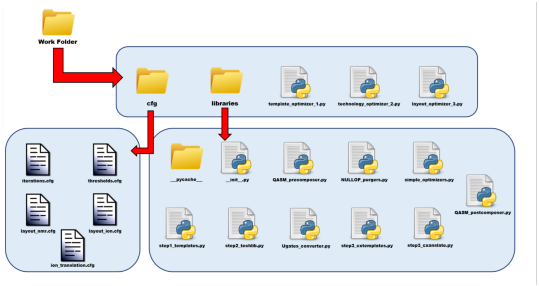


Figure 3.6: Overview of the project implemented in [15]

Chapter 4

Machine Learning

4.1 Introduction

In this chapter, the theoretical background and related works for Deep Learning architectures are introduced. The focus is on models for graph-structured data and probabilistic generative models, concluding with applications to quantum circuit optimization.

Graph Neural Networks (GNN)

Graph Neural Networks (GNN)[16] are a class of deep learning algorithms specifically designed for operations on graph-structured data. The fundamental principle behind GNN is Neural Message Passing, an iterative procedure in which each node aggregates neighbors information; Figure 4.1 reports an example of a neighborhood for a node.

In this work, GraphSAGE[17], an inductive variant of GNN, is used, allowing generalization of information not provided during training.

Graph Attention Networks (GAT)

While GNNs aggregate messages without assigning them a weight, Graph Attention Networks (GAT), introduced in [18], embed a self-attention mechanism to dynamically weight the importance of every connection. For the aim of this project, GATs become relevant for the discrimination of high-density gates (e.g. multi-qubit gates).

Gated Recurrent Units (GRU)

Gated Recurrent Units (GRU)[19] are the evolution of Long Short Term Memory (LSTM)[20] for efficient sequence modeling, solving the problem of vanishing gradient.

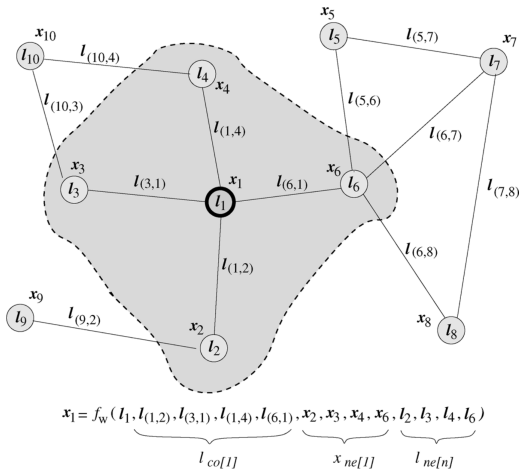


Figure 4.1: Graph and the neighborhood of a node. The state x1 of the node 1 depends on the information contained in its neighborhood.[16]

4.2 Variational Autoencoder

The Variational Autoencoder (VAE) is a generative model introduced in [21]. A VAE works by learning probabilistic mapping, which is the main difference from a standard autoencoder described in [22], which learns a deterministic mapping, to model the probability distribution $p(x)$ of a dataset X.

The fundamental challenge for VAE is to analytically calculate the probability distribution $p(z|x)$, namely the probability of having a certain latent vector z given x as input. The latter definition is commonly known as the intractability of the posterior distribution, which arises in the Bayes Theorem:

$$p(z|x) = \frac{p_\theta(x|z)p(z)}{p(x)} \tag{4.1}$$

where:

- $p_\theta(x|z)$ is the *likelihood*. Its defines the reconstruction loss: its maximum is equivalent to a minimum in the Mean Squared Error (MSE), formulated in Equation 4.6, between the input and the reconstruction.
- $p(z)$ is the *prior* distribution of the latent variables.
- $p(x)$ is the *evidence* or marginal likelihood, representing the probability of observing the data averaged over all possible latent configurations.

Calculating $p(x)$, requires marginalizing all the possible configurations of z, that end up with the integral:

$$p(x) = \int p(x|z)p(z)dz \tag{4.2}$$

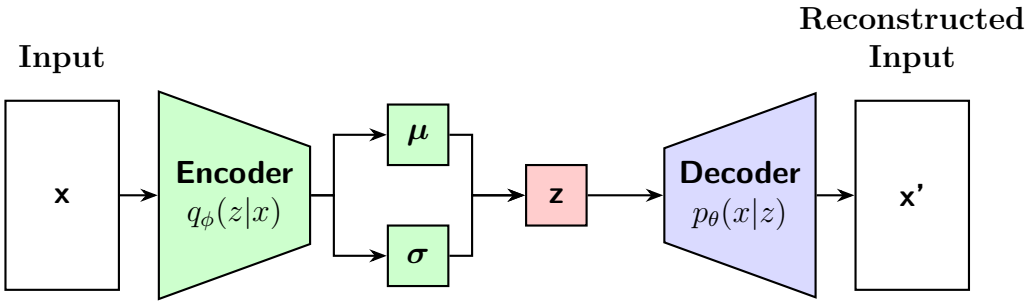


Figure 4.2: Visual representation for a Variation Autoencoder: **Encoder** and **Decoder** are the common terms to refer respectively to the approximate distribution and likelihood.

which is non-linear (does not have analytical solution) and computationally complex requiring exponential time to be solved directly. To overcome this, variational inference has been introduced, approximating the true posterior distribution with a simpler parametrized distribution $q_\phi(z|x)$. The objective is to maximize the Evidence Lower Bound (ELBO) derived as follows:

$$\mathcal{L}(\theta, \phi; x) = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}(q_\phi(z|x)||p(z)) \tag{4.3}$$

where:

- $\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)]$ is the reconstruction term. How well the parametrized distribution $q_\phi(z|x)$ reconstructs x from z
- $D_{KL}(q_\phi(z|x)||p(z))$ is the Kullback-Leibler divergence, acting as a regularization term, forcing the learned latent distribution close to the prior $p(z)$

A critical innovation in VAEs is the reparameterization trick. Instead of sampling the latent vector directly from the output of the parametrized distribution $q_\phi(z|x)$, z is redefined as:

$$z = \mu + \sigma \odot \epsilon, \text{ where } \epsilon \sim \mathcal{N}(0,1) \tag{4.4}$$

this makes the sampling process differentiable with respect to μ and σ, allowing backpropagation through the sampling layer. The overall architecture for a Variation Autoencoder is illustrated in Figure 4.2.

4.3 Directed Acyclic Graph Variational Autoencoder

Building on the definition of a VAE, many models have been developed specifically for a particular data structure. In [23], a Variational Autoencoder for Directed

Acyclic Graphs (D-VAE) is introduced. DAGs can model many real-world problems, such as the architecture of a neural network, connection structures of Bayesian networks, even in electronic circuit design, realizing a target function is a DAG optimization task. In this context, it is crucial to find novel efficient methods for DAG optimization. The novel VAE architecture introduces a different approach for message passing; instead of passing all the neighbors' messages simultaneously, they propose an **asynchronous message passing scheme** to encode the computation on DAGs, respecting the computation dependencies among the nodes. With this scheme the model is able to map the discrete input space of DAGs into a continuous latent space, in which every DAG has its unique embedding.

The architecture is divided into encoder and decoder as standard VAE:

- *Encoder*: an update function $\mathcal{U}$ is used to compute the hidden state of each node based on its neighbors as in Figure 4.3; this step ensures the following of the topological order of DAGs, which is crucial for propagation of its original structure through the layers. The encoding step can be seen as a Gated Recurrent Unit (GRU) layer.
- *Decoder*: another GRU is used to perform the same asynchronous message passing scheme of the encoder; aggregating all the predecessors, the DAG is constructed node by node, sampling the next one based on the neighbors until the ending node is reached. Edges are predicted according to the current node.

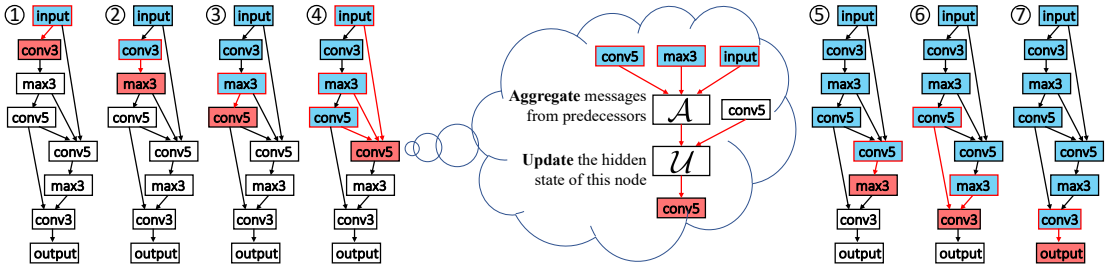


Figure 4.3: Encoding procedure in [23]; the central cloud clearly explains how the hidden state is computed for a particular node

4.4 DVAE for quantum circuit optimization

As stated in subsection 3.1.1, the use of DAGs for quantum circuit representation is widespread, a prominent example is Qiskit (section 3.2). For this reason, it is logical to merge quantum circuit optimization with VAE designed for this versatile data structure (section 4.3).

In [5], AltGraph is proposed. It is presented as a robust approach to generate valid equivalent quantum circuits. The pipeline followed by AltGraph starts with

converting the original quantum circuit into a DAG; subsequently the generative graph model reconstructs a new DAG, which is converted back to a quantum circuit; the process is depicted in Figure 4.4.

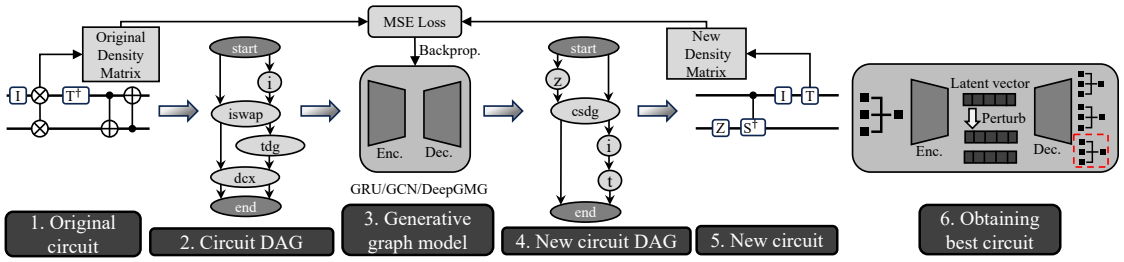


Figure 4.4: Overview of AltGraph process for generation of quantum circuits [5]

AltGraph follows the structure of the model proposed in [23], embedding crucial modifications for quantum circuits. An important step in VAE implementation is the design of the loss function; in this specific architecture, the total loss function is calculated combining the KL-divergence of the encoder, defined for two continuous probability distribution over the same probability space by the integral:

$$D_{KL}(P||Q) = \int_{-\infty}^{\infty} p(x) \ln \frac{p(x)}{q(x)} dx \tag{4.5}$$

and the Mean Squared Error (MSE) of the reconstructed circuit's density matrix with respect to the original one.

The MSE is an estimator of the average squared difference between the estimated values and the actual values, formulated for the matrices ρ and $\hat{\rho}$ as:

$$\mathrm{MSE}(\rho, \hat{\rho}) = \frac{1}{N^2} \sum_{i=1}^{N} \sum_{j=1}^{N} |\rho_{ij} - \hat{\rho}_{ij}|^2 \tag{4.6}$$

The graph generation then follows simple but necessary rules to ensure that the reconstructed DAG preserves its acyclicity and that the correspondent quantum circuit is valid:

- *Number of Qubits.* The output density matrix must have the same size of the input one.
- *Gate and Gate Connections.* Predict the exact number of connections for the current node.
- *Circuit Types.* To reduce the complexity of the task, a reduced number of quantum gates is used (subsection A.1.1).

The database used for AltGraph training and testing is composed of randomly generated quantum circuits with a maximum of 6 qubits per circuit and a maximum of 192 gates per circuit. The whole framework is tested with different generative models, [5] reports very promising results in terms of gate and depth reduction. For further details on this approach see subsection A.1.2.

4.5 Reinforcement learning

Reinforcement Learning (RL) is a subfield of machine learning focused on agents which take actions in an environment with the aim of maximizing a reward, a function accurately designed to grow when the agent takes the right sample. The learner does not know which actions to take, but it must discover which actions yield the best reward by trying them [24].

The problem is mathematically described as a Markov Decision Process (MDP), defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where:

- $\mathcal{S}$, state space
- $\mathcal{A}$, action space
- $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0,1]$ state transition probability function
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, reward function
- $\gamma \in [0, 1]$ is the discount factor to weight future rewards.

At each step, the agent selects an action that observes the current state. As a consequence, the agent receives a scalar reward and transitions to a new state. The aim of the agent is to find an optimal policy that maximizes the expected return.

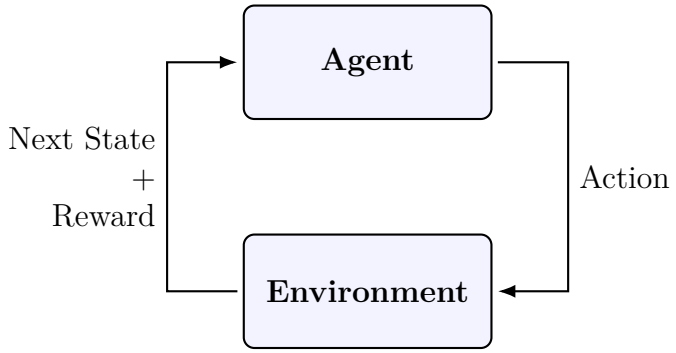


Figure 4.5: The standard Reinforcement Learning interaction loop. The agent influences the environment via actions and receives feedback in the form of states and rewards.

4.6 Comparison of Quantum Circuit Optimization Frameworks

In this section, Table 4.1 reports the main optimization solutions adopted by the different frameworks described previously. In particular, we can see that while Qiskit and t|ket> rely on deterministic, rule-based heuristics for DAG and ZX-calculus transformations, AltGraph introduces a probabilistic approach via D-VAE. By leveraging latent embeddings and a loss function led strategy (KL and MSE), AltGraph enables global structural discovery, transitioning from local gate reduction rules to a data driven exploration of quantum circuit equivalences.

Feature	**Qiskit (Transpiler)**	**t|ket>**	**AltGraph (D-VAE)**
Primary Method	Rule-based / Heuristic	Peephole / Graph Rewrite	Generative / Probabilistic
Representation	DAG / QASM	ZX-Calculus / DAG	DAG (Latent Embedding)
Optimization Goal	Depth / Gate Count	Phase Gadgets / T-count	Structural Discovery
Hardware Awareness	High (Calibration data)	High (Architecture-led)	Emerging (Loss-function led)

Table 4.1: Main optimization solutions adopted by the different frameworks.

Part II

PANZEROTTO

Chapter 5

Compiler structure

In this chapter the architecture PANZEROTTO is presented, a GenAI model for quantum circuit optimization; the approach is based onto a DAG-VAE designed for the creation of a reduced dimension latent distribution and for the generation of optimized versions of the input quantum circuits, aiming for a trade-off between fidelity and circuit length. The following discussion is strictly demonstrative, without including the code implementation.

All the details needed to understand the code are reported in Appendix C.

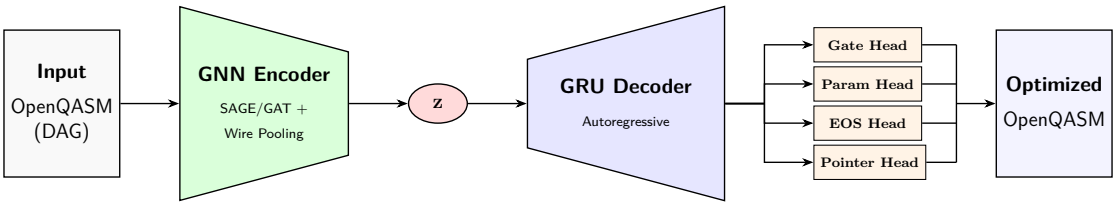


Figure 5.1: The PANZEROTTO architecture. The model employs a GNN-based encoder with Wire Pooling to form the latent space z. The autoregressive GRU decoder then feeds four specialized heads, whose outputs are assembled into the final optimized OpenQASM circuit.

5.1 Offline Processing and Data Representation

The input circuit is assumed to be an OpenQASM 2.0[25] file. The input file must be transformed into a suitable data structure for a neural network. As previously discussed in subsection 3.1.1, DAG structure is the standard de facto for modern optimization techniques.

5.1.1 From QASM to DAG

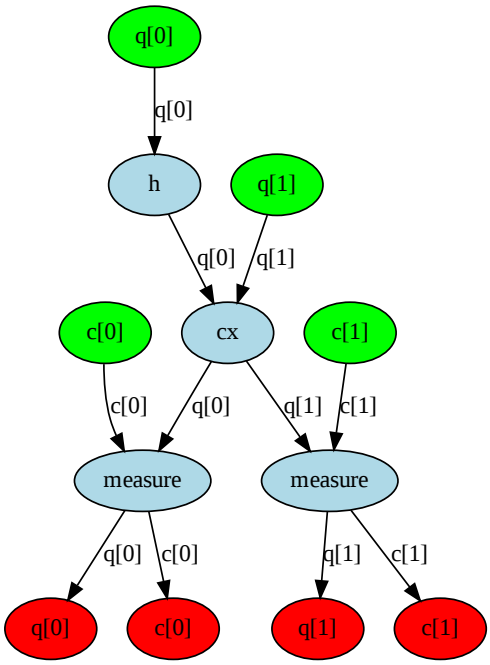
QASM code is parsed and converted in a directed acylcic graph (DAG), where each gate becomes a node in the graph, while the qubits create the graph's edges. From now on, the terms `gate` and `node` are used interchangeably.
The same equivalence goes for `qubit` and `edge`.

An example of conversion in depicted in Figure 5.2.

OpenQASM 2.0

```
OPENQASM 2.0;
include "qelib1.inc";
qreg q[2];
creg c[2];
h q[0];
cx q[0], q[1];
measure q -> c;
```

(a) Code snippet



(b) DAG Representation obtained with Qiskit's DAG drawer.

Figure 5.2: Visual comparison of circuit representations.

5.1.2 Feature Extraction

Neural Networks work on lists of features extracted from the original data. Providing a large number of information to the network is important to facilitate the work done by the model for learning the structure of the data that compose the dataset.

In this work, a list of features is associated with every node and every edge.

Following the extraction feature method proposed by [26], the features of the nodes are constructed as three concatenated lists:

- *One Hot Encoding*: One-Hot vector that identifies the type of quantum gate with respect to a predefined dictionary (see section B.1).
- *Gate Parameters*: rotational parameters normalized in the range $[-\pi, \pi]$, padded to zero for non-parametric gates.
- *Arity & Topology*: respectively number of involved qubits and fan-in/fan-out.

Also edge's features list is a concatenation of vectors, in particular:

- *Role*: One-Hot vector that identifies the role for the qubit (control, target or single)
- *Qubit Index*: One-Hot vector that identifies which qubit the edge represents.

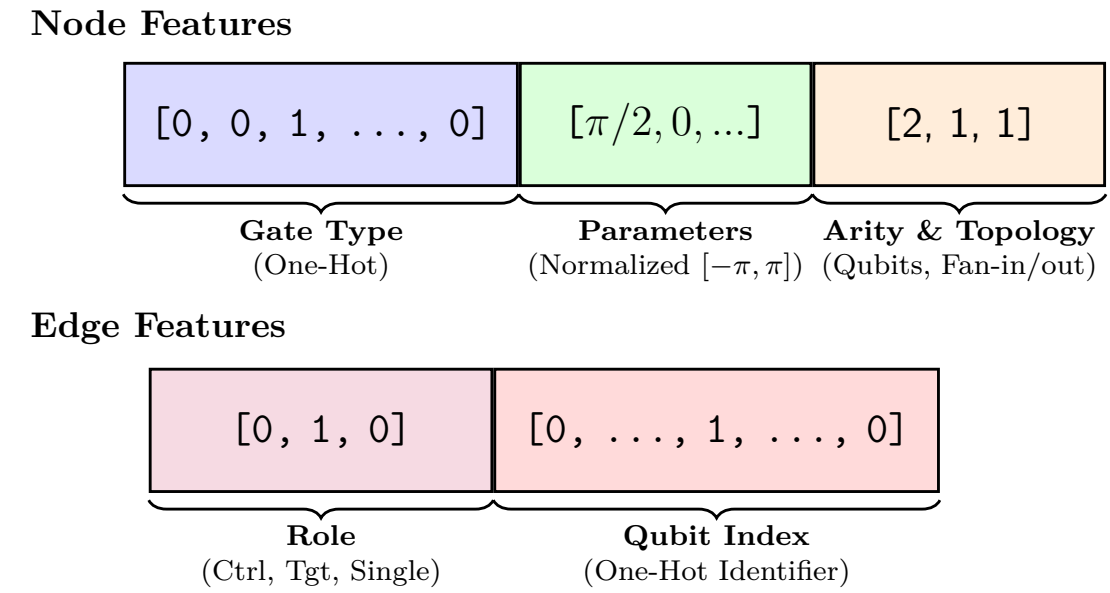


Figure 5.3: Visual representation of the Feature Extraction process. The raw DAG attributes are encoded into concatenated vectors. Top: Node features including gate type, parameters, and topology. Bottom: Edge features encoding the qubit role and index.

5.1.3 Post-processing

The output of the decoder is a sequence of information that is re-parsed into OpenQASM 2.0. A filter is applied to ensure the validity of the generated gates.

5.2 Model

The core of the work is a Variational Autoencoder with encoding performed via a GNN and decoding based on a autoregressive GRU with a pointer network. The use of Deep Learning generative models for quantum circuit optimization is supported by [5], which demonstrates the effectiveness of this type of architectures.

The architecture of PANZEROTTO underwent a significant transformation. This section analyzes this evolution across the three main components: the refinement of Encoder, the transition to an Autoregressive Decoder, and the redefinition of the Loss Function to incorporate non-differentiable reward metrics.

The encoding module was initially designed to transform DAGs into latent representations. The evolution focused on increasing the information collected during feature extraction, specifically regarding edge attributes and hardware constraints. The very first Encoder focused on basic feature extraction from DAGs, implementing ordinal encoding for gate types to create initial tensor representations. Later, the Encoder was refactored to better handle variable graph sizes. A critical change is the replacement of standard GCN layers with SAGEConv layers, allowing for more robust message passing. Concurrently, an Edge Multilayer Perceptron (MLP) was introduced to recognize edge attributes (e.g. control or target), enabling the model to weigh neighboring nodes based on their connectivity roles. The last important change was to specific "wire embeddings" for qubits, adding a modulation mechanism to adjust node features dynamically based on neighbors weights.

As for the encoding step also the decoding phase evolved during the work, starting from a simple sequence generator that pretty soon revealed to be insufficient to build complex data structures as DAG are. One of the first improvements was to add a mechanism of Autoregressive Sampling to generate gates sequentially, ensuring that the generation of the current step depended on the history of previously generated gates. Moreover, to reduce the generation of invalid circuits, qubits sampling was adapted to generate an output based on *gate arity*, namely ensuring that the number of sampled qubits matched the specific gate type being generated. To improve the fidelity of the output, a "Best-of-N" sampling strategy was introduced, generating multiple candidate circuits sampling multiple times the latent space and selecting the optimal one.

The initial loss function was rooted in standard VAE metrics, combining reconstruction error with a KL divergence term to regularize the latent space. To balance exploration and regularization during training, entropy and temperature scaling were added to the sampling process within the loss calculation. A reward mechanism was introduced with the use of Reinforcement Learning to optimize for circuit quality rather than just reconstruction accuracy. An attempt of using a differentiable statevector fidelity is worth mentioning, it would have allowed backpropagation without the use of RL but was abandoned due to complexity of introduction of another framework into the pipeline[27].

5.2.1 Final Model

The final architecture of PANZEROTTO integrates a hybrid GNN encoder utilizing SAGEConv layers and wire embeddings for feature extraction, an autoregressive GRU decoder constrained by gate arity and Best-of-N sampling, and a multi-objective loss function that augments traditional VAE metrics with Reinforcement Learning rewards for non-differentiable metrics.

Encoder

The encoder has the role of compressing the structural information of the circuit to create a continuous latent space, nevertheless propagating all the information needed to reconstruct the circuit. The architecture is composed of layers of Graph Neural Networks (GNN), in particular GraphSAGE or GAT, or an alternation of both, based on D-VAE principles proposed in [23] to efficiently handle the asynchronous message passing needed for DAG structures.

A key feature of this encoding process is the **Wire Pooling** mechanism. While standard global pooling works by aggregating all the nodes to create a final representation of the graph, Wire Pooling aggregates nodes based on the qubit they operate on. This mechanism creates a set of *Wire Embeddings* that can be seen as a "memory" for each qubit.

A final global pooling is still performed to put all the information together.

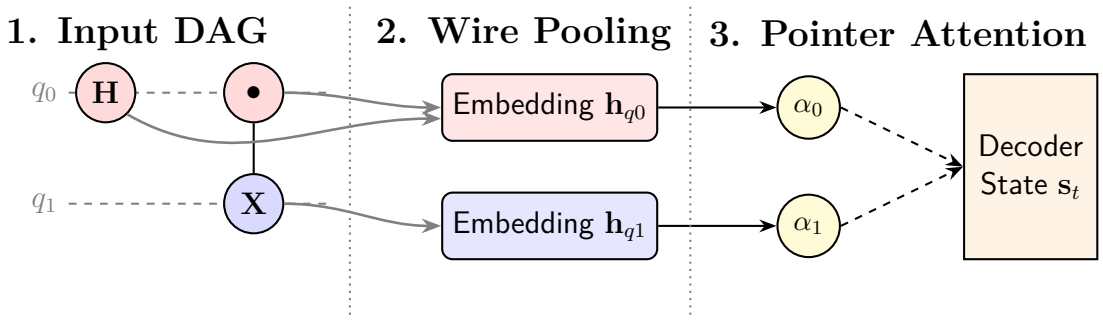


Figure 5.4: Visual explanation of the Wire Pooling mechanism and Pointer Head. Gate features on the same wire (q_0) are aggregated into a single embedding, which is then compared with the decoder state.

Decoder

The decoding procedure is based on a Gated Recurrent Unit (GRU). The core idea is to reconstruct the circuit node by node. Since a quantum circuit has multiple parameters to be taken into account, the decoder is created by putting together 4 different heads, each one with a specific role:

- *Gate Head*: this head samples the current gate from a predefined set section B.1.
- *Parameters Head*: this head predicts gate parameters with a Gaussian policy.
- *EOS Head*: is needed to sample a particular kind of node called End Of Signal (EOS), useful to prematurely terminate generation allowing actual optimization in terms of gate count.
- *Pointer Head*: this head is repeated as many times as the maximum number of qubits per gate is reached, is in project it is assumed to be 3. Each Pointer Head samples a possible qubit calculating a similarity score between the current state of the decoder and the wire embedding created by the encoder.

Loss Function

The training aim to minimize a multi-objective loss function, balanced between reconstruction and KL regularization following a typical VAE:

$$\mathcal{L} = w_{KL} D_{KL} + w_g L_g + w_p L_p + w_q L_q \tag{5.1}$$

where L_g,L_p,L_q are, respectively, gate, parameters, and qubit losses added with their own weight and calculated from the decoder prediction with respect to the encoder embedding. In particular:

- L_{gate} is calculated as a Categorical Cross Entropy loss, aimed at minimizing the distance between the predicted distribution probability and the One-Hot encoding of the initial graph.
- $L_{parameters}$ is calculated as a Smooth L1 loss for the prediction of continuous parameters.
- L_{qubit} is calculated as a masked Categorical Cross Entropy loss, to tackle the multiclass classification problem only on the qubits that are used by the current gate.

Chapter 6

Compiler Workflow

In this chapter the whole operative workflow of the compiler is described. Starting from dataset generation, through training procedure, ending with inference strategy for optimization.

6.1 Dataset Generation

The dataset has been generated collecting quantum circuits from well known repositories [28][29][30][31], focusing solely on real algorithms, avoiding the generation of randoms circuits since the philosophy behind this work is to propose a tool for real world applications suitable for quantum computing execution.

Throughout a pre-processing step, each circuit is:

1. Excluded if the number of qubits bigger than the hardware backend dimension (in this case fixed at 18, but can be modified accordingly to any backend).

2. Excluded if the number of gates is greater than a constant. This step can be avoided, but for experimental purposes it is better to find a threshold corresponding to the 95 percentile of the whole dataset.

3. Cleaned from auxiliar structures defined in the OpenQASM language that are not useful for the description of the circuit (barriers, measures, id).

4. Encoded following the procedure detailed in subsection 5.1.2.

A tensor dataset "*.pt*" is created to be read by the neural network. The final dataset results in a list of circuits with associated information (gate count, depth, statevector, number of qubits) useful for training.

6.2 Training Strategy: Hybrid VAE-RL

The training procedure has been divided into two distinct steps; the philosophy behind this decision can be summarized as "**First learn, then optimize**".

1. "*First learn*" is translated in a supervised training of the VAE, focused on learning the basics of "grammar" for quantum circuits and creating a model able to reconstruct any quantum circuit with maximum fidelity.

2. "*then optimize*" refers to the second step in which the pre-trained model is fine-tuned with a reinforcement learning approach focused on the maximization of a Reward Function based on structural metrics of the circuit.

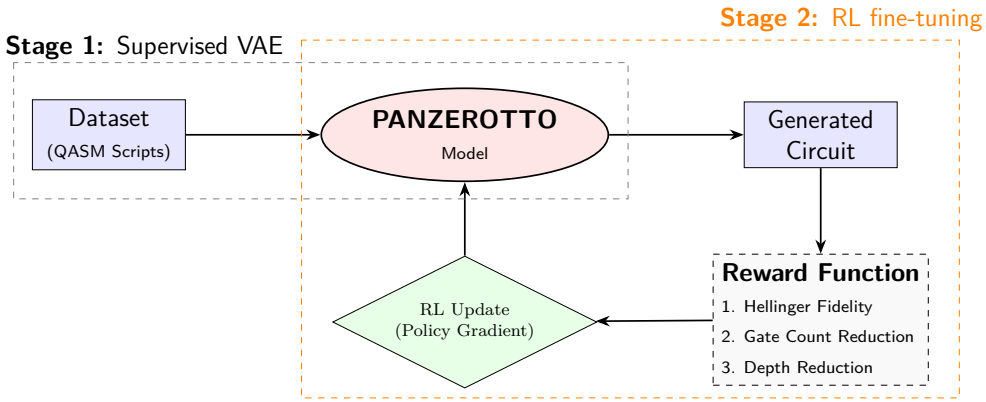


Figure 6.1: The hybrid training workflow. Stage 1 focuses on circuit reconstruction. Stage 2 uses Reinforcement Learning to optimize non-differentiable metrics.

6.2.1 Supervised VAE

To manage the complexity of the latent space created by a large variety of quantum circuits, two adaptive mechanisms have been adopted.

Two-stage curriculum learning

During the development of the system, an issue arises. Indeed, as in Figure 8.1, the loss term for the generation of qubits is significantly greater than the other terms. Two-stage training is used to overcome this problem; in the first stage, qubit loss is heavily weighted to prioritize wire routing, and in the second stage, the weights are balanced to optimize the total loss function.

Temperature Annealing

To manipulate exploration, a temperature parameter is applied to the output distribution, changing its shape. While an high temperature ($T > 1$) flattens the probability distribution to allow the model wider exploration, a lower temperature ($0 < T < 1$) sharpens the distribution limiting exploration.

6.2.2 Reinforcement Learning Fine-Tuning

In quantum circuit optimization, the core aim is to reduce the total number of quantum gates in a circuit to avoid decoherence. It is crucial to keep the fidelity high, otherwise the algorithm represented by the circuit will be totally changed. Gate count, depth and fidelity are non-differentiable metrics, meaning that if embedded in the loss function of a standard supervised VAE prevent backpropagation of the gradient, breaking the training process. One way to insert non-differentiable variable in a training loop is Reinforcement Learning. The reward is derived from the fidelity of the generated circuit with respect to the original one, calculated as the Hellinger Fidelity (assuming projective measurements, the phase is ignored), depth and gate count reduction.

$$Reward = w_F \cdot Fidelity + w_{depth} \cdot \Delta Depth + w_{gc} \cdot \Delta GateCount \tag{6.1}$$

The total loss function used in the training procedure is:

$$\mathcal{L}_{total} = \alpha \mathcal{L}_{VAE} - \beta \mathcal{L}_{RL} \tag{6.2}$$

6.3 Inference

Once a trained model is obtained, it works like a stochastic compiler.

N samples are generated with respect to the input circuit, sampling the latent space multiple times, obtaining the following:

$$\{C'_1, C'_2, ..., C'_N\} \tag{6.3}$$

Each candidate is valued with respect to the original circuit. The circuit C'_{best} is accepted only if it satisfies a customizable acceptance policy. The latter must be well designed, e.g. if only circuits with fidelity equal to 1 (maximum) are accepted, the model will not always give an optimized version of the circuit because the latent space will be restricted, reducing optimization capabilities, or, e.g. accepting lower fidelity circuits, much more optimization options are possible.

Part III

Results

Chapter 7

Experimental Setup

In this chapter, the computational framework employed to perform the model trainings is detailed. Given the high complexity of the quantum circuits under investigation, specifically circuits with up to 16 qubits, the use of High-Performance Computing (HPC) resources was mandatory to achieve the following results.

7.1 HPC Infrastructure Overview

The simulations were executed on the **LEGION** cluster, managed by *HPC@POLITO* at Politecnico di Torino[32]. The architecture is based on a high-bandwidth and low-latency RDMA network (Infiniband).

7.1.1 Hardware Specifications

The computational experiments were performed on the *gpu_v100_ext* nodes. The hardware configuration for each node is summarized in Table 7.1:

Component	Specification
CPU	2 x Intel Xeon 6130 (32 cores)
RAM	384 GB DDR4
GPU	4 x NVIDIA V100 SXM 32 GB

Table 7.1: HPC node specifications

7.1.2 Software Stack and Numerical Libraries

A list of tools and libraries utilized in the framework.

- Distributed Data Parallel (DDP) [33] for automated parallel execution of the training procedure across multiple nodes and GPUs.

- Qiskit statevector simulator.
- Qiskit Aer simulator.
- AMARETTO, for accelerated statevector simulation[34], integrated in the pipeline as in Figure 7.1.

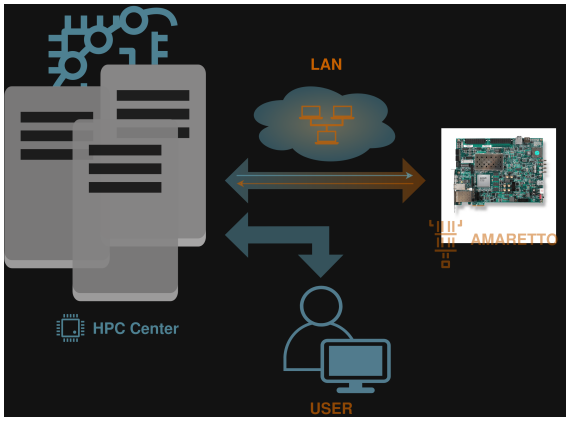


Figure 7.1: Use of AMARETTO integrated in a HPC workflow

7.1.3 Runtime setup

The following results have been generated by running multiple model trainings. The usage of HPC resources was mostly allocated on GPU nodes of the cluster. The typical setup is composed of five nodes with four GPUs per node, with a total of 20 GPUs. Small adjustments were made during tests in accordance with the current status of the cluster.

Chapter 8

Training Curves

8.1 Figures configuration

This chapter reports the training curves for different initial datasets. The title of the figure reports the type of encoding procedure followed for that experiment, the maximum number of qubits in the dataset, and the total number of circuits in the dataset. The caption reports a detailed description of the behavior.

Each figure reports six different plots, respectively:

- **Total Loss**: behavior of the Total Loss Function in epochs, calculated as in section 5.2.1.
- Overview of the components of the Loss Function
- Detail of **Gate Loss**
- Detail of **KL Divergence**
- Detail of **Parameter Loss**
- Detail of **Qubit Loss**

The following results are based solely on **Step 1**, as in Figure 6.1, of the PANZEROTTO framework. This choice will be cleared later after the presentation of the results.

8.2 General behavior of training results

In the experiments, the Total Loss is characterized by an exponential decay within the first 100 epochs, followed by an almost linear decay up to the end of training. This trend suggests that, after a quick embedding of the global features of the

circuits, the model tries to balance the loss components to reconstruct as best as possible the input dataset.

Of particular interest is the Parameter Loss; it starts at the beginning with a lower value with respect to the others, stopping at a threshold much closer to 0 with respect to the previous threshold. This behavior suggests that the model is better suited to learn continuous parameters rather than discrete sampled Gates of Qubits.

Gate Loss and Qubit Loss still exponentially decay in the first epochs, stabilizing at the end to a threshold value greater than 0.

KL Divergence increases in the first epochs, stabilizing later around a constant value. This suggests a stable encoding technique.

The main problem reported by these plots is that the **Qubit Loss never drops below the value of 1.0**, as in Figure 8.1, this is a systematic critical issue, signaling a marked difficulty for the model in handling complex classification tasks such as Gate prediction, suggesting that the network is trapped in a local minimum where it converges to a uniform probability distribution across the gate set.

For each experiment, the reported number of epochs corresponds to the highest epoch count reached before divergence was observed.

The *asymmetric learning hierarchy* of the model clearly highlights the limits of this approach for Quantum Circuit Optimization. The discrete nature of quantum operators requires superior discriminative capacity. Final considerations on model limitation are reported at the end of this part after further investigations through Principal Component Analysis (PCA).

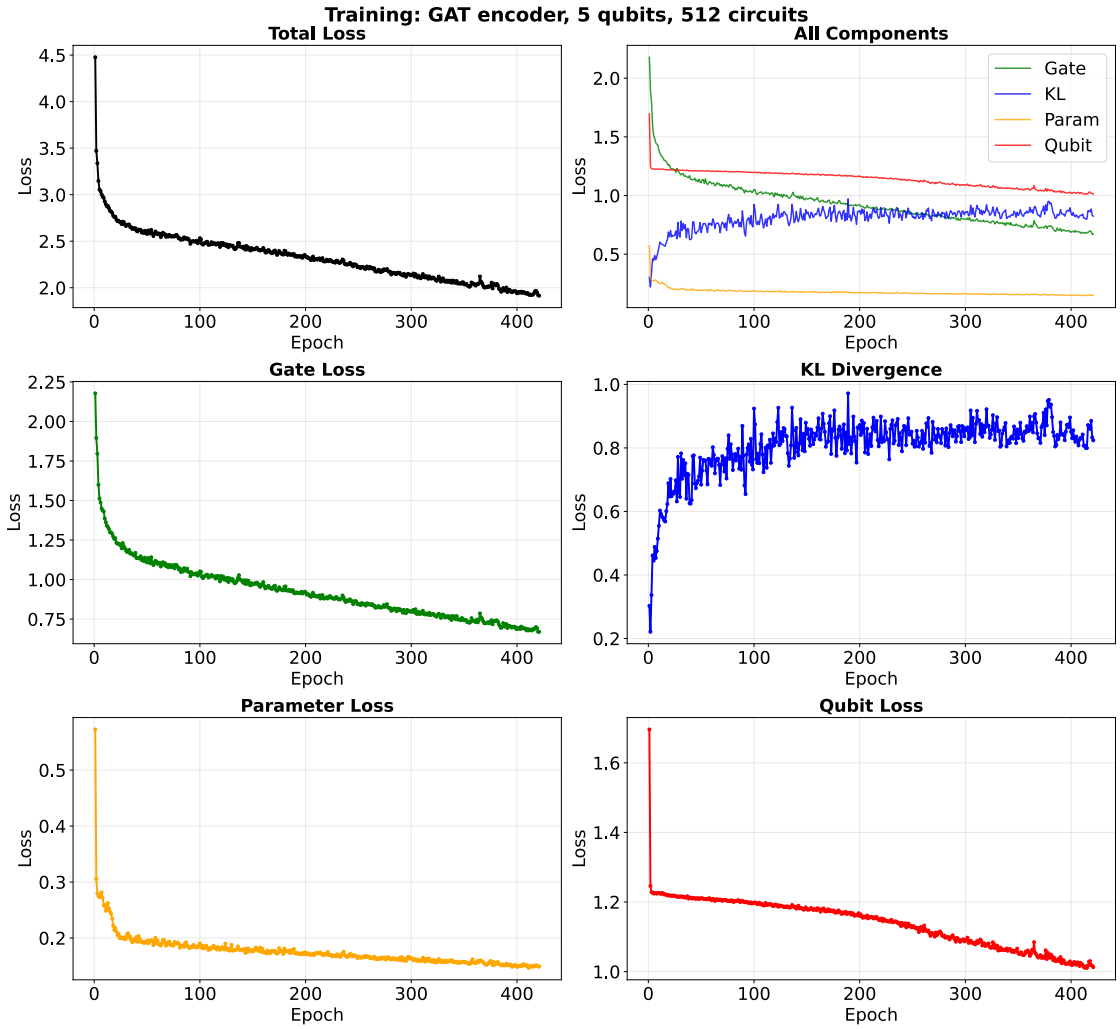


Figure 8.1: Training of the model employing a GAT encoder, evaluated on 512 quantum circuits with a maximum of 5 qubits. By extending the training to 400 epochs, a steady convergence is observed across all reconstruction metrics. The Gate and Qubit losses reach approximately 0.70 and 1.05 respectively. The oscillation in the KL divergence beyond epoch 100 reflects the ongoing competition between the latent space regularization and the fine-tuning of structural reconstruction.

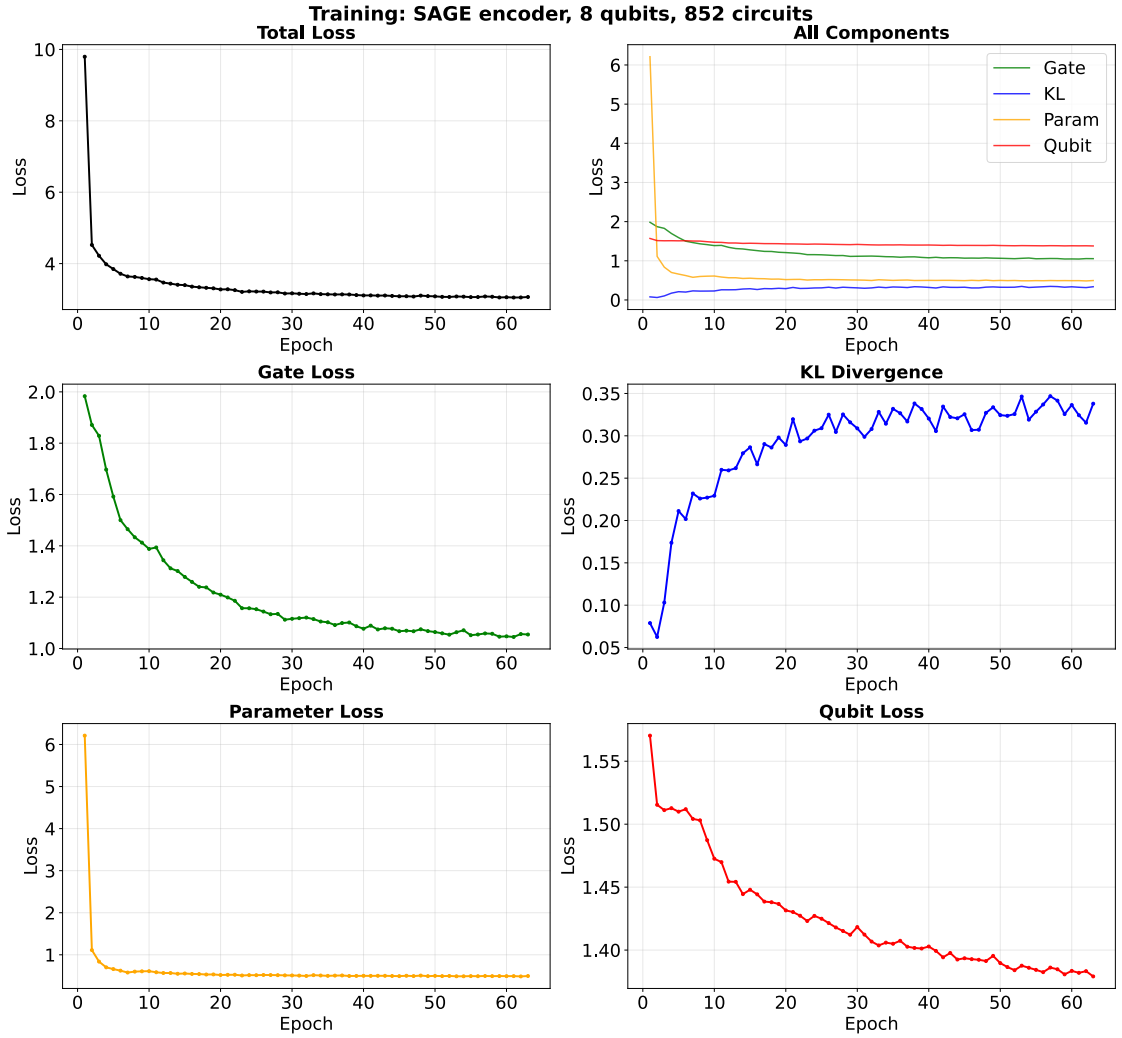


Figure 8.2: Training performance and loss decomposition for the model configured with a GraphSAGE encoder, utilizing a dataset of 852 quantum circuits with a maximum of 8 qubits. The analysis tracks the convergence of the total loss alongside the individual reconstruction components: gate classification, parameter regression, and qubit mapping. The GraphSAGE architecture demonstrates a rapid initial descent in parameter loss, while the KL divergence exhibits a characteristic logarithmic growth, indicating a stable regularization of the latent distribution.

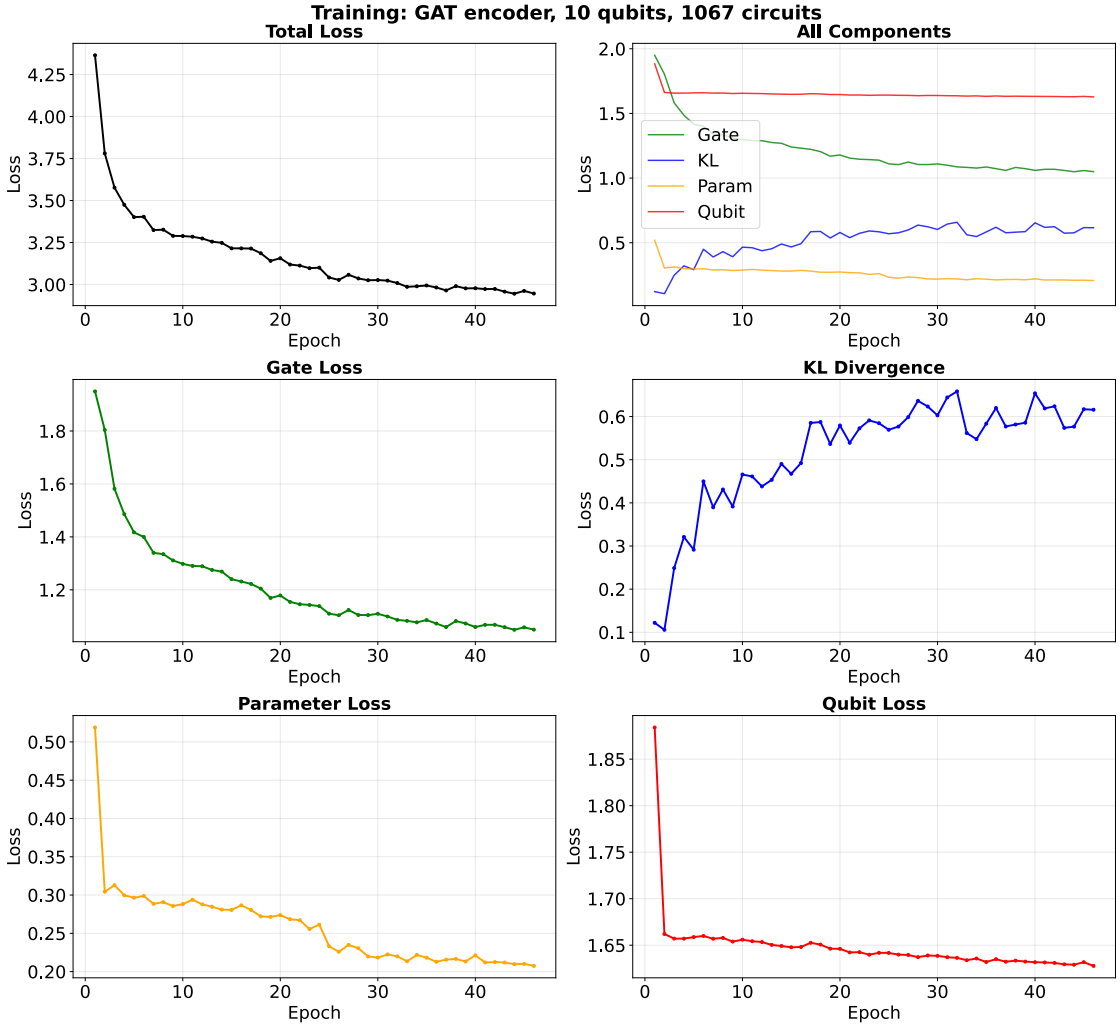


Figure 8.3: Convergence analysis of the model training process utilizing a GAT encoder for a dataset of 1067 circuits with a 10 qubit limit. The individual plots illustrate the evolution of the total loss and its constituent components: gate classification loss, parameter regression loss, qubit allocation loss, and the KL divergence. The results demonstrate a consistent reduction in total loss, though the qubit and gate components exhibit an asymptotic trend, suggesting the onset of a performance ceiling for this specific encoder architecture.

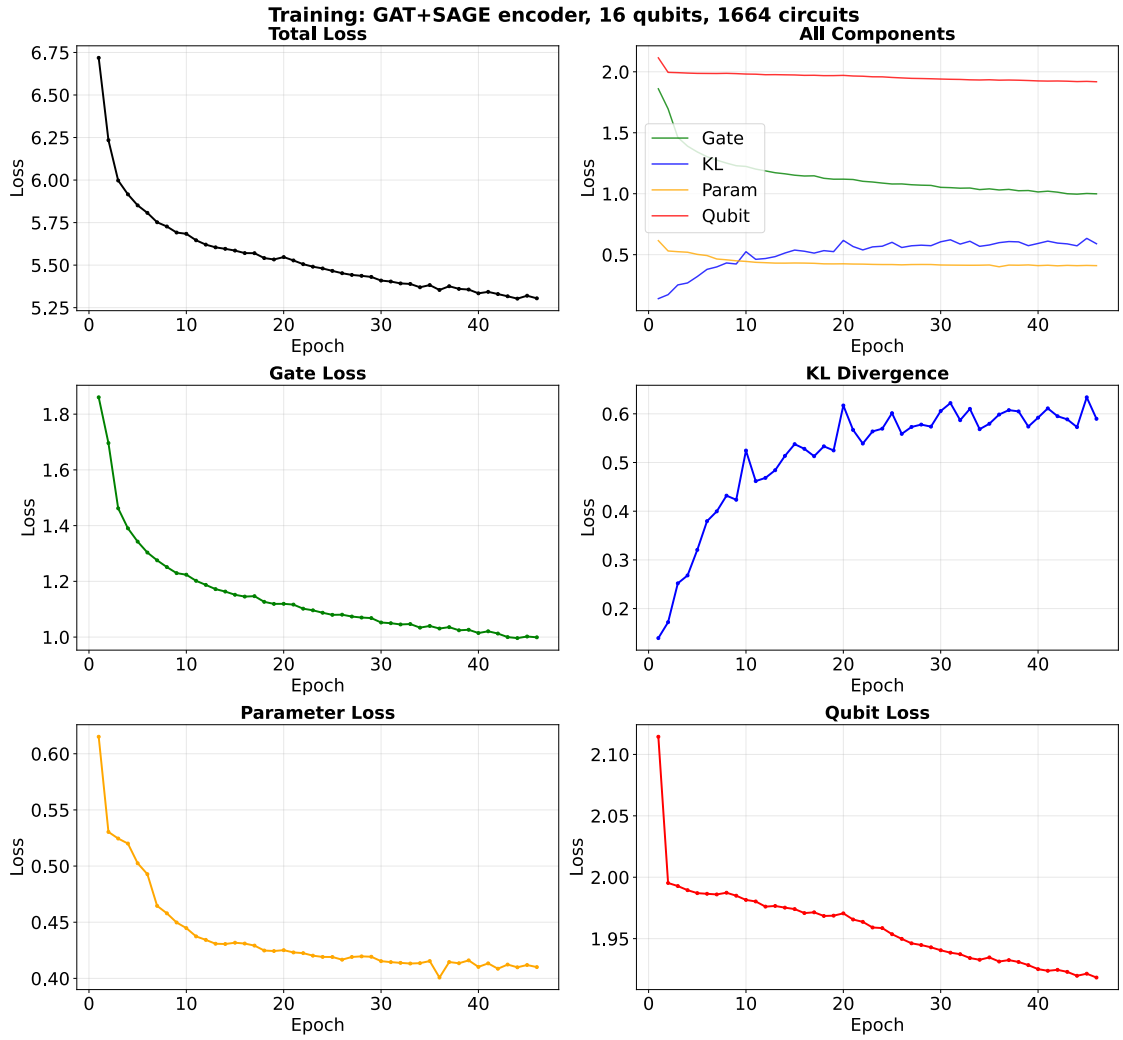


Figure 8.4: Training for the mixed GAT+SAGE encoder configuration on a database with a maximum of 16 qubits in 1664 circuits. Left to right, top to bottom: Total Loss, combined loss components, Gate Loss, KL Divergence, Parameter Loss, and Qubit Loss. The steady increase in KL divergence suggests the expansion of the latent space representation, though reconstruction accuracy for discrete gate operations shows limited improvement after epoch 30.

Chapter 9

Principal Component Analysis (PCA)

The primary objective of employing PCA in this framework is to perform a geometric assessment of the latent space generated by the GNN encoder (specifically, the GAT and mixed GAT+SAGE architectures). While the empirical loss curves discussed in the previous chapter provide a macroscopic metric of the network's optimization trajectory, they ignore the high-dimensional topological structure of the learned representations.

In the training procedure the encoder tries to map the discrete topological structure of quantum circuits into a continuous, high-dimensional vector space. The relative distances and clustering of these state vectors in the latent space affect the model's capacity to decode and discriminate between different quantum gates, continuous parameters, and qubit targets. To analyze this, PCA is utilized to project the high-dimensional graph embeddings onto a lower-dimensional subspace maximizing the variance of the projected data.

The dynamical sampling of PCA projections across different training epochs provides a direct visualization of the feature extraction process. In a theoretically optimal training scenario, the projections should evolve from a random distribution at the initial epochs into a highly structured manifold where circuits with similar structures and properties aggregate in different sections of the latent space to form well-separated clusters.

The persistent inability of the model to reduce the **Qubit Loss** below the 1.0 threshold, as in section 8.2 suggests a topological overlap in the latent space. By observing the temporal evolution of the PCA projections across the 5, 8, 10, and 16 qubit experiments, it is possible to confirm whether the network fails to separate the structural representations of different types of quantum circuits in the initial dataset.

This analysis is necessary to further investigate the model behavior when trained

on different initial datasets. The following sections report the PCA visualizations at specific epoch intervals for each dataset configuration, illustrating the progressive structuring of the model's latent space before divergence occurs.

9.1 Figures configuration

The figures in this chapter follow a predefined structure. In each figure the X-axis (Component 1) and Y-axis (Component 2) represent the leading orthogonal eigenvectors. These directions capture the maximum scalar variance of the dataset when projected from the high-dimensional latent space into a 2D plane.

To evaluate exactly which quantum circuit properties drive the formation of clusters, the PCA is repeated four times: three panels apply specific color mappings to the data points, the fourth panel is reserved as an objective control baseline.

- **Top-Left Panel (Num Qubits):** This plot employs a linear color gradient to map the total number of qubits allocated in each circuit. The purpose of this panel is to visualize whether the neural network embeds information based on the dimension of the quantum register in the circuits.
- **Top-Right Panel (Depth):** This plot utilizes a base-10 logarithmic color scale to represent the depth of the quantum circuit. Logarithmic scaling is necessary to normalize the wide variance in circuit complexity, aimed at highlighting how the network handles shallow versus deep circuits.
- **Bottom-Left Panel (Num Gates):** This panel also uses a logarithmic color scale to display the total gate count of the circuit. Discriminating in this case how the network handles short versus long circuits.
- **Bottom-Right Panel (Structural Baseline):** This is the control plot, rendering all embedded circuits in a uniform grayscale.

9.2 Overview of PCA results

The analysis of the plots reveals a critical architectural limitation. Within the exact same spatial section, where circuit depth is linearly separable, data points associated with a different number of qubits exhibit severe chromatic overlapping. The network fails to form discrete, segregated clusters for the circuits as a function of their quantum register dimensionality. This inability to separate the structural representations of different types of quantum circuits provides a direct demonstration of the model asymmetric learning hierarchy. The encoder compresses the data into a continuous structure that forces the amalgamation of topologically distinct architectures. In general, no visible cluster appears with any of the proposed colorings.

Moreover, the examination of the grayscale panel results in a continuous density distributions of the embedded points, lacking the distinct cluster fragmentation required for classification tasks.

This geometric evidence correlates with the results derived from the training curves: the persistent inability of the model to reduce the Qubit Loss below the 1.0 threshold is confirmed by the topological overlap in the latent space.

The observation of the PCA projections suggests that the model systematically fails in the microscopic and macroscopic reconstruction of the quantum circuit topology.

9.3 Five qubits experiment

The temporal observation of PCA projections across random training epochs reveals a geometric evolution of the latent embeddings. Initially, the network projects the circuits into a "V" shaped structure, as in Figure 9.1. As training progresses, the latent space strictly resolves into a structured, multipolar manifold, characterized by distinct orthogonal branches forming a cross-like topology, as in Figure 9.2 and Figure 9.3. Towards the end (Figure 9.4) model divergence occurs, scattering the points in a high-entropy distribution.

9.4 Eight qubits experiment

Increasing the dataset dimension to 852 circuits and changing the encoding network from GAT to SAGE, the data are initially partially clustered according to the number of gates in the circuits, as in Figure 9.5 (bottom-left panel), then collapsing to a "V" shape manifold without any visible clustering, as in Figure 9.6.

9.5 Ten qubits experiment

With a dataset of 1067 circuits and GAT encoding the plots assume multiple shapes without any visible cluster, as in Figure 9.7, Figure 9.8 and Figure 9.9.

9.6 Sixteen qubits experiment

The highest complexity dataset incorporates 1664 circuits up to 16 qubits, processed via a mixed GAT+SAGE encoder architecture. The PCA trajectory for this configuration demonstrates a critical representational breakdown. Unlike structured arcs or orthogonal branches of lower-dimensional sets, the 16-qubit latent

space initially collapses into an isotropic point mass, as in Figure 9.10 and Figure 9.11, and subsequently fractures into highly compressed disjointed lines, as in Figure 9.12 and Figure 9.13.

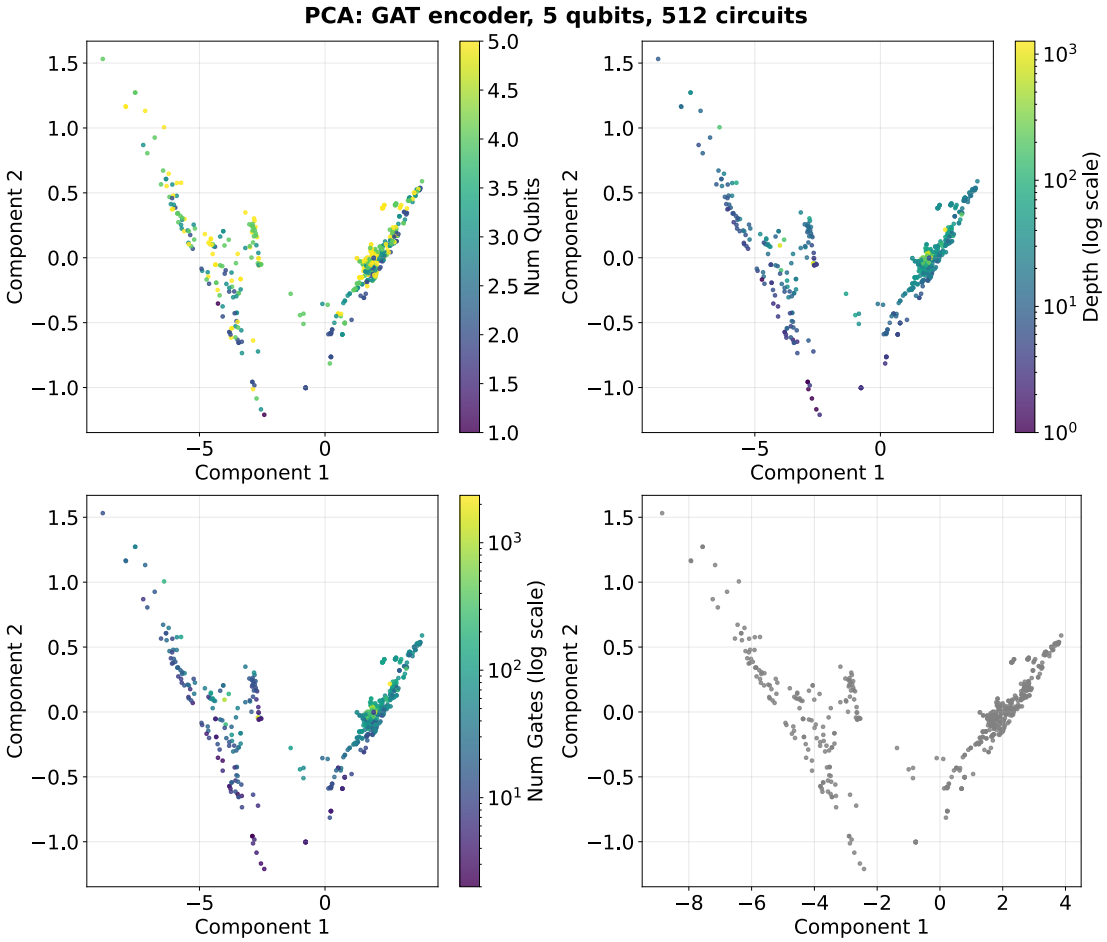


Figure 9.1: Two-dimensional PCA of the latent space generated by the GAT network. The embeddings represent a dataset comprising 512 distinct quantum circuits, constrained to a maximum quantum register dimensionality of 5 qubits, extracted at training epoch 1.

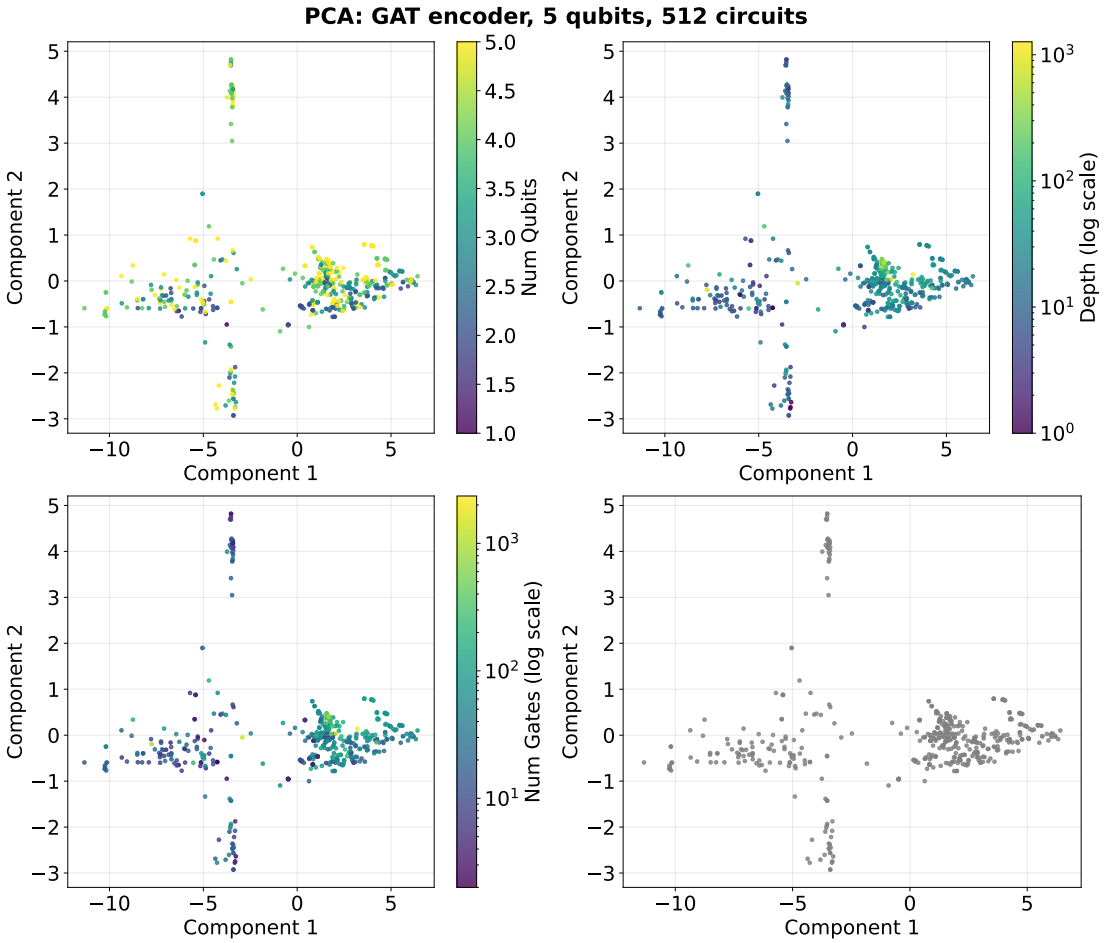


Figure 9.2: Two-dimensional PCA of the latent space generated by the GAT network. The embeddings represent a dataset comprising 512 distinct quantum circuits, constrained to a maximum quantum register dimensionality of 5 qubits, extracted at training epoch 10.

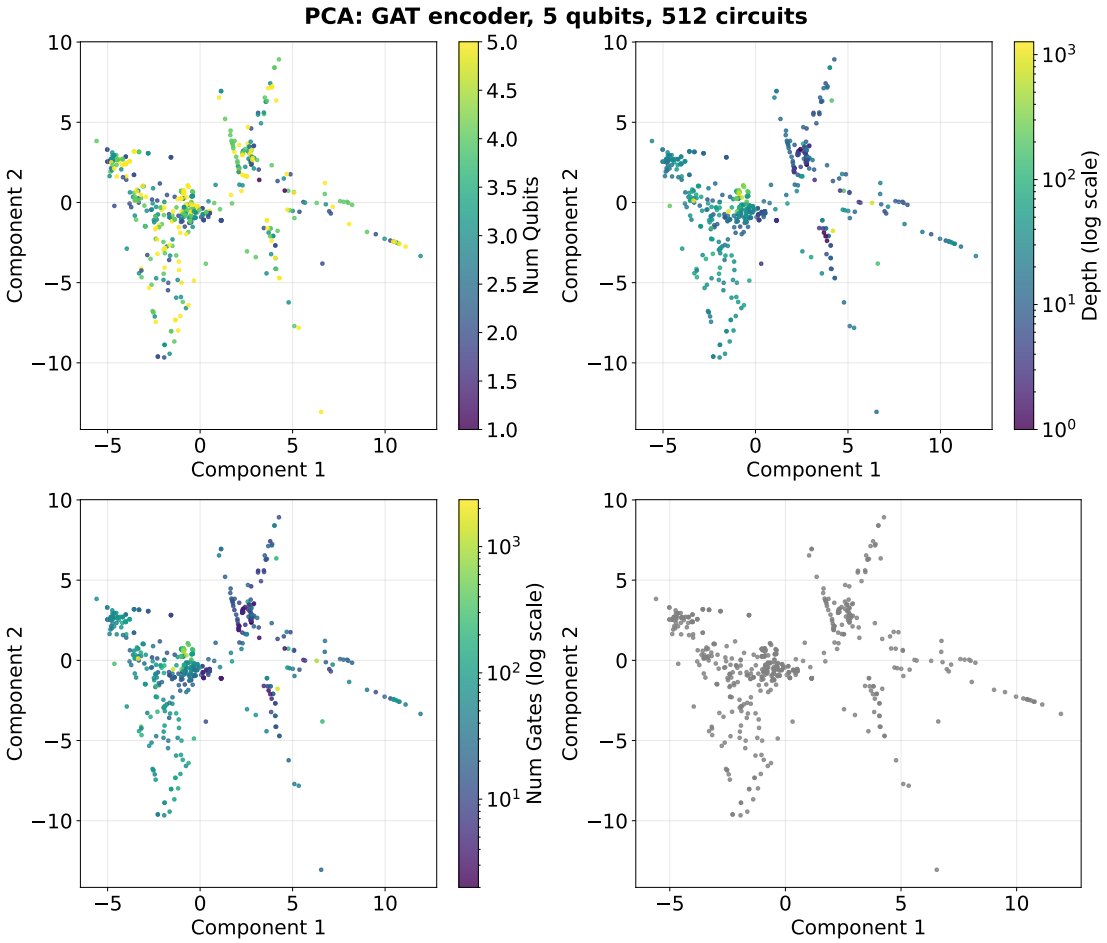


Figure 9.3: Two-dimensional PCA of the latent space generated by the GAT network. The embeddings represent a dataset comprising 512 distinct quantum circuits, constrained to a maximum quantum register dimensionality of 5 qubits, extracted at training epoch 100.

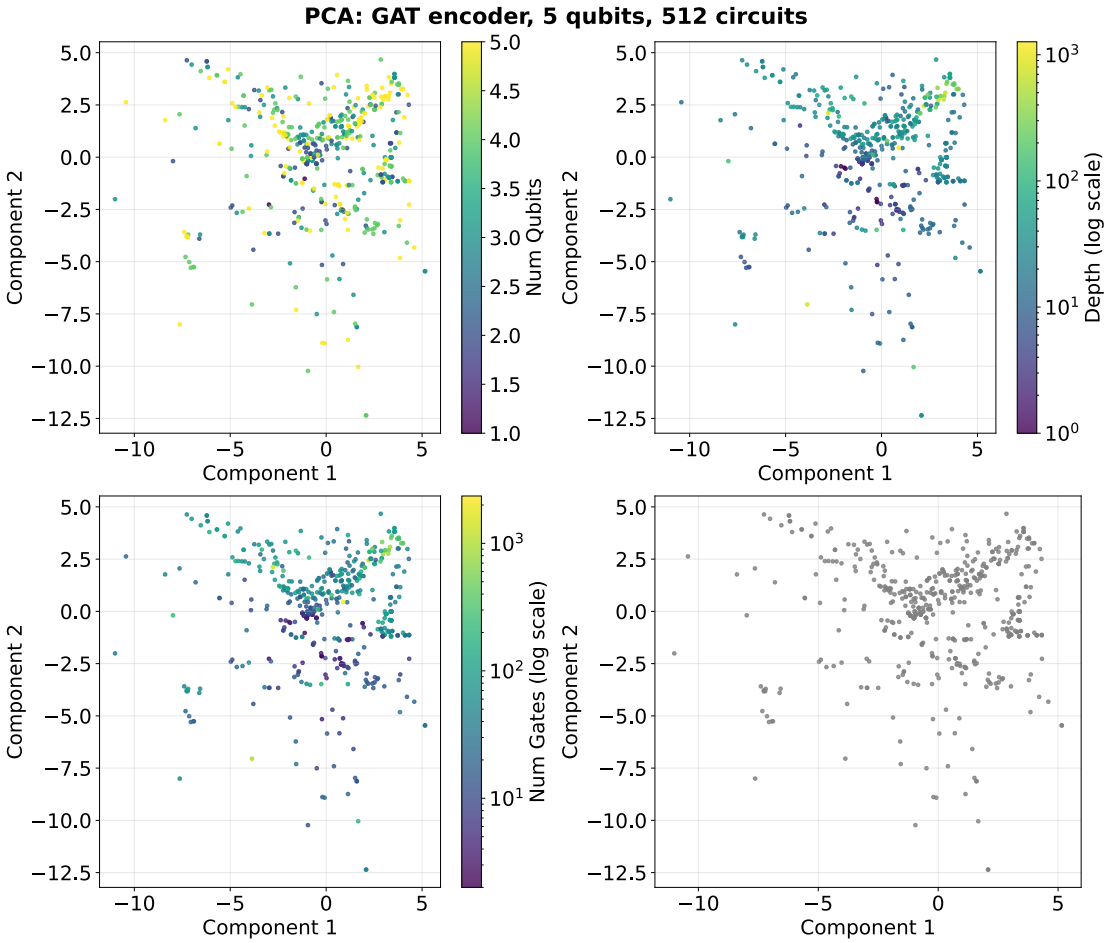


Figure 9.4: Two-dimensional PCA of the latent space generated by the GAT network. The embeddings represent a dataset comprising 512 distinct quantum circuits, constrained to a maximum quantum register dimensionality of 5 qubits, extracted at training epoch 420.

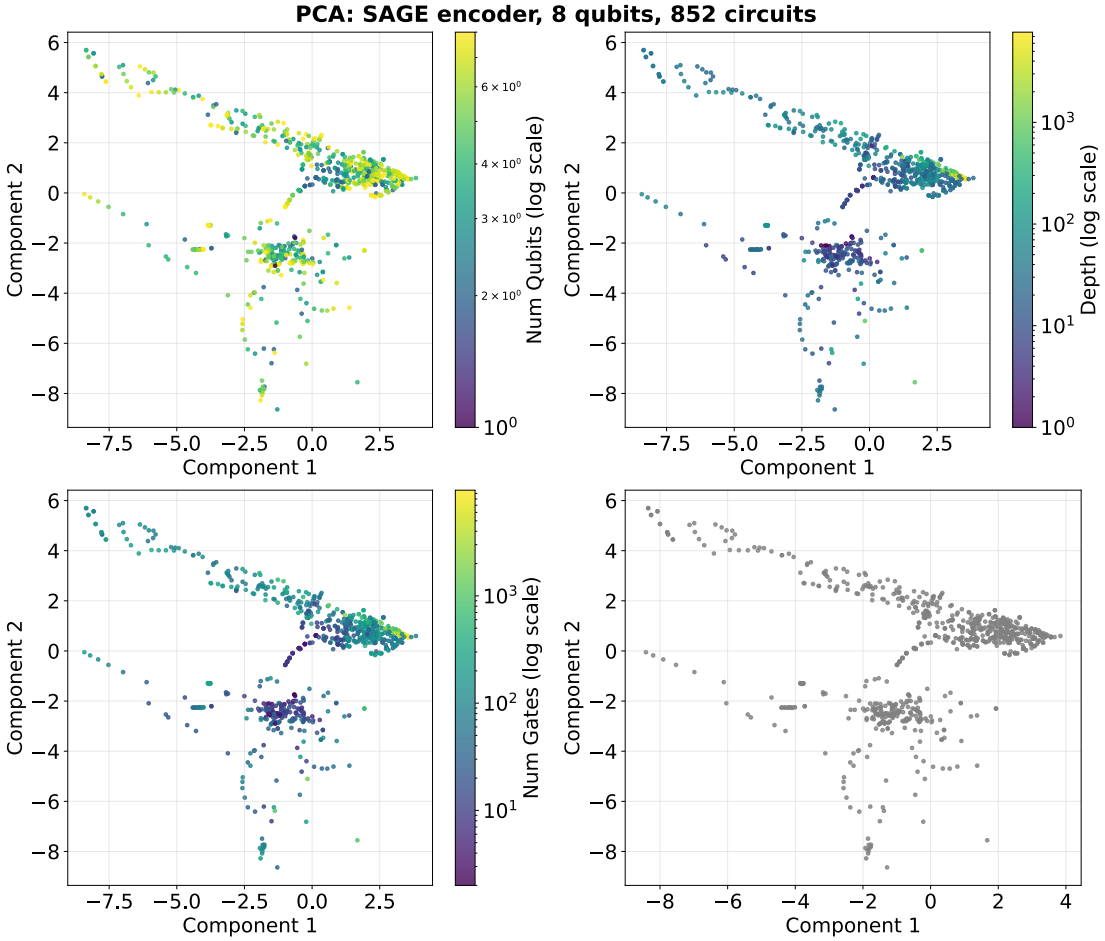


Figure 9.5: Two-dimensional PCA of the latent space generated by the SAGE network. The embeddings represent a dataset comprising 852 distinct quantum circuits, constrained to a maximum quantum register dimensionality of 8 qubits, extracted at training epoch 10.

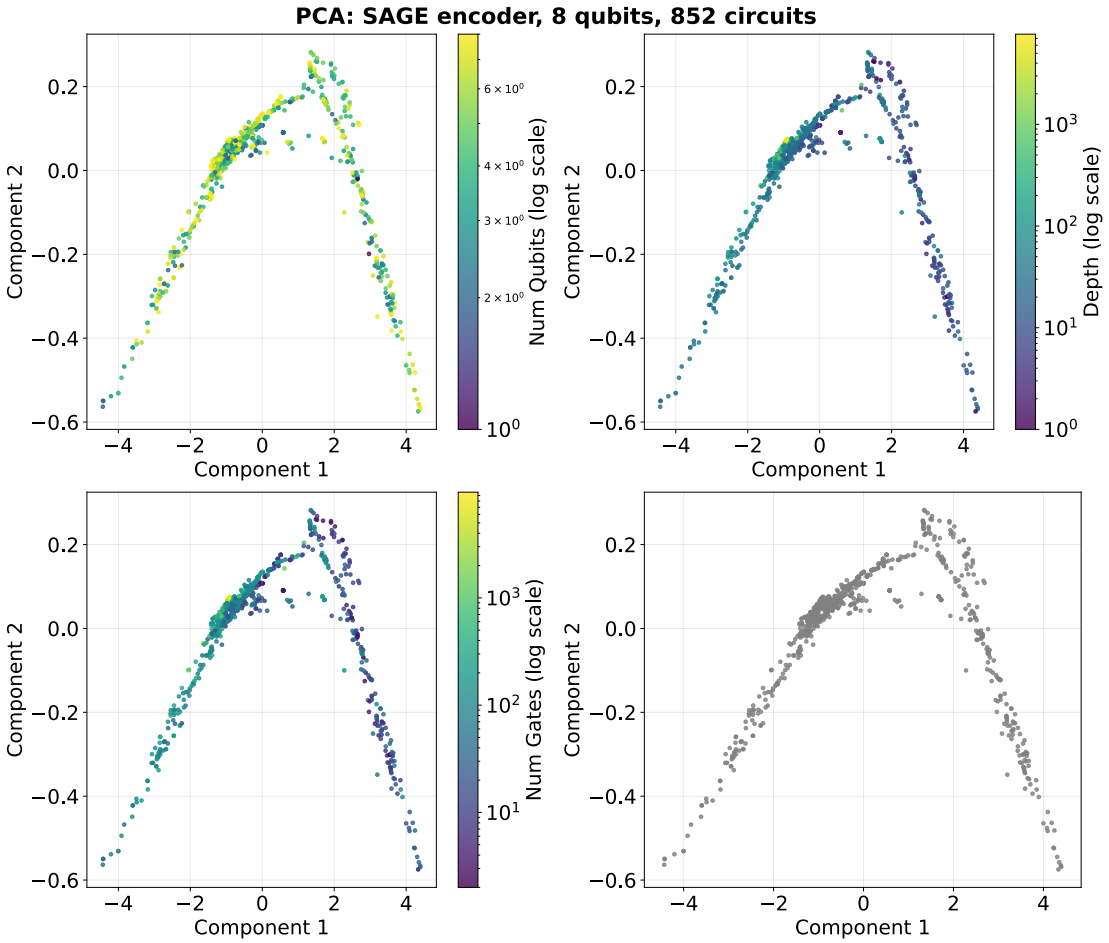


Figure 9.6: Two-dimensional PCA of the latent space generated by the SAGE network. The embeddings represent a dataset comprising 852 distinct quantum circuits, constrained to a maximum quantum register dimensionality of 8 qubits, extracted at training epoch 55.

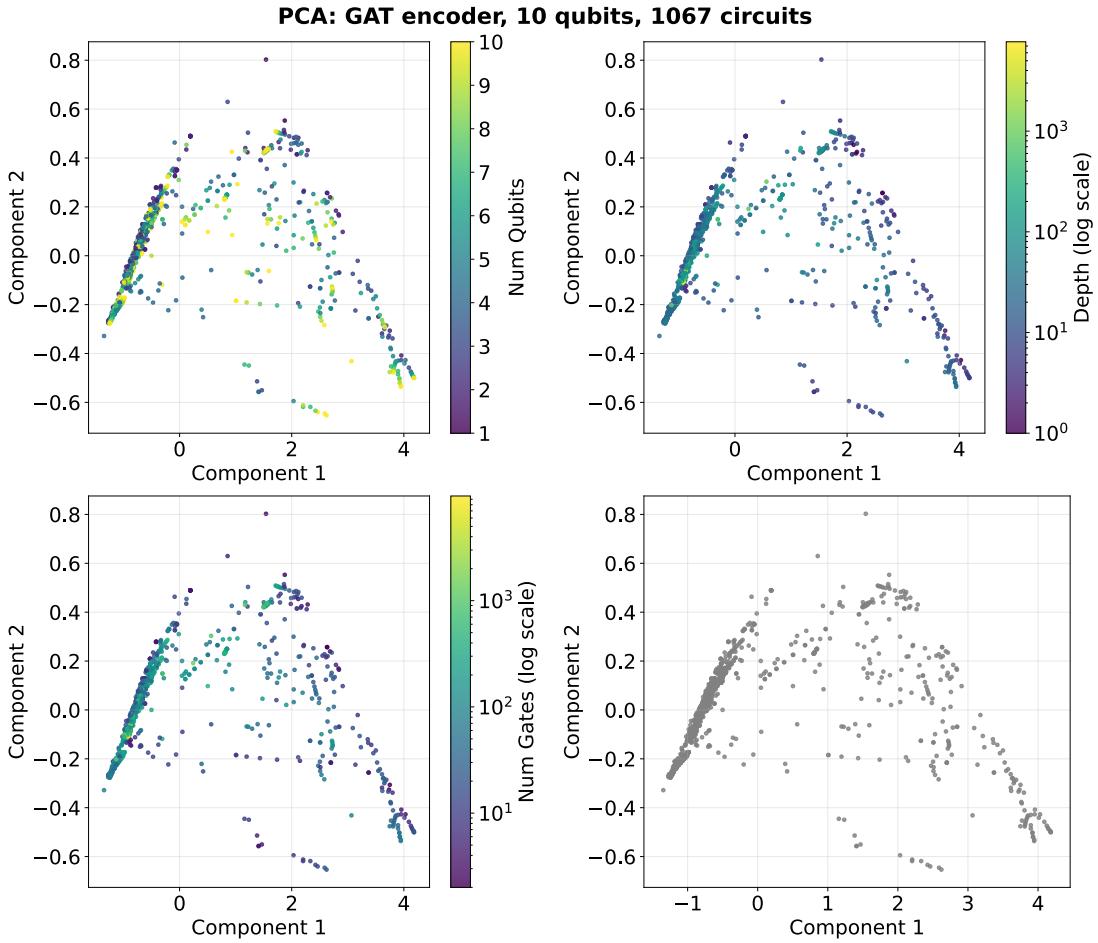


Figure 9.7: Two-dimensional PCA of the latent space generated by the GAT network. The embeddings represent a dataset comprising 1067 distinct quantum circuits, constrained to a maximum quantum register dimensionality of 10 qubits, extracted at training epoch 2.

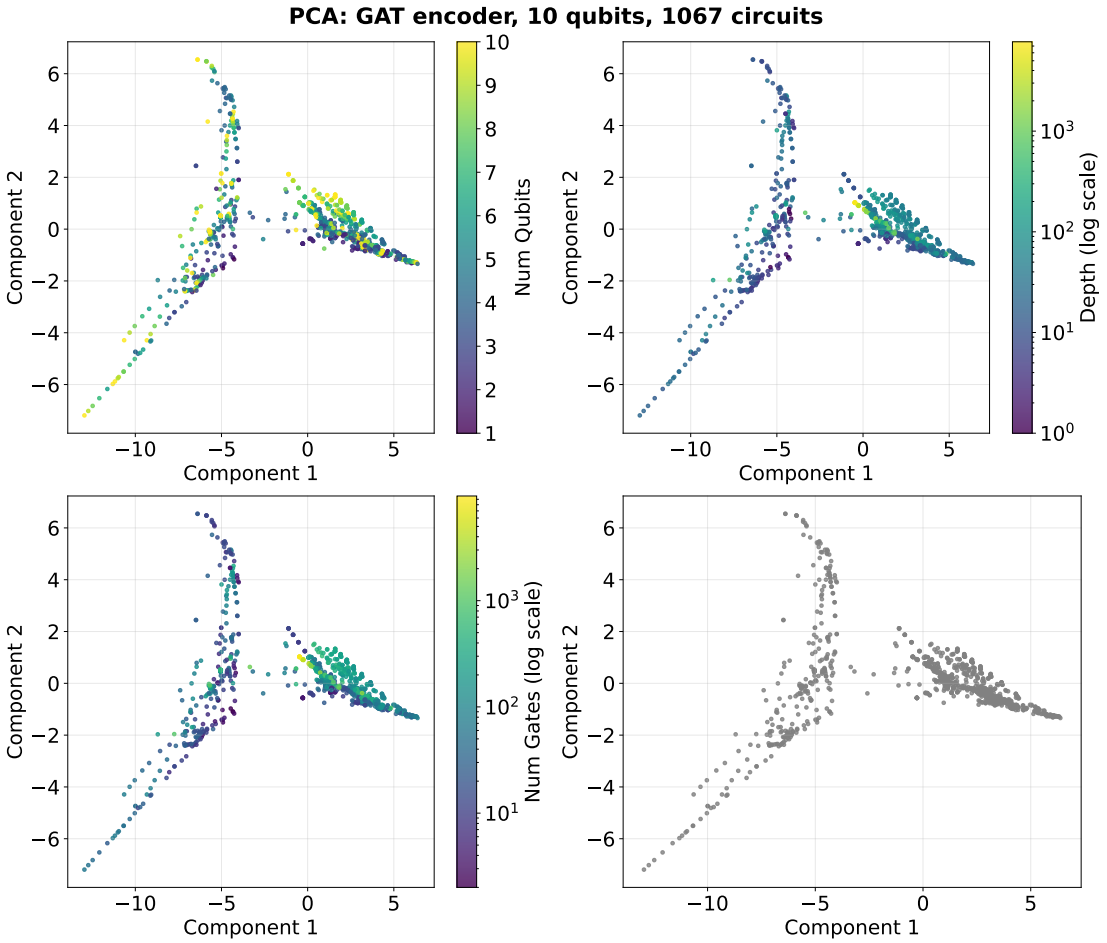


Figure 9.8: Two-dimensional PCA of the latent space generated by the GAT network. The embeddings represent a dataset comprising 1067 distinct quantum circuits, constrained to a maximum quantum register dimensionality of 10 qubits, extracted at training epoch 10.

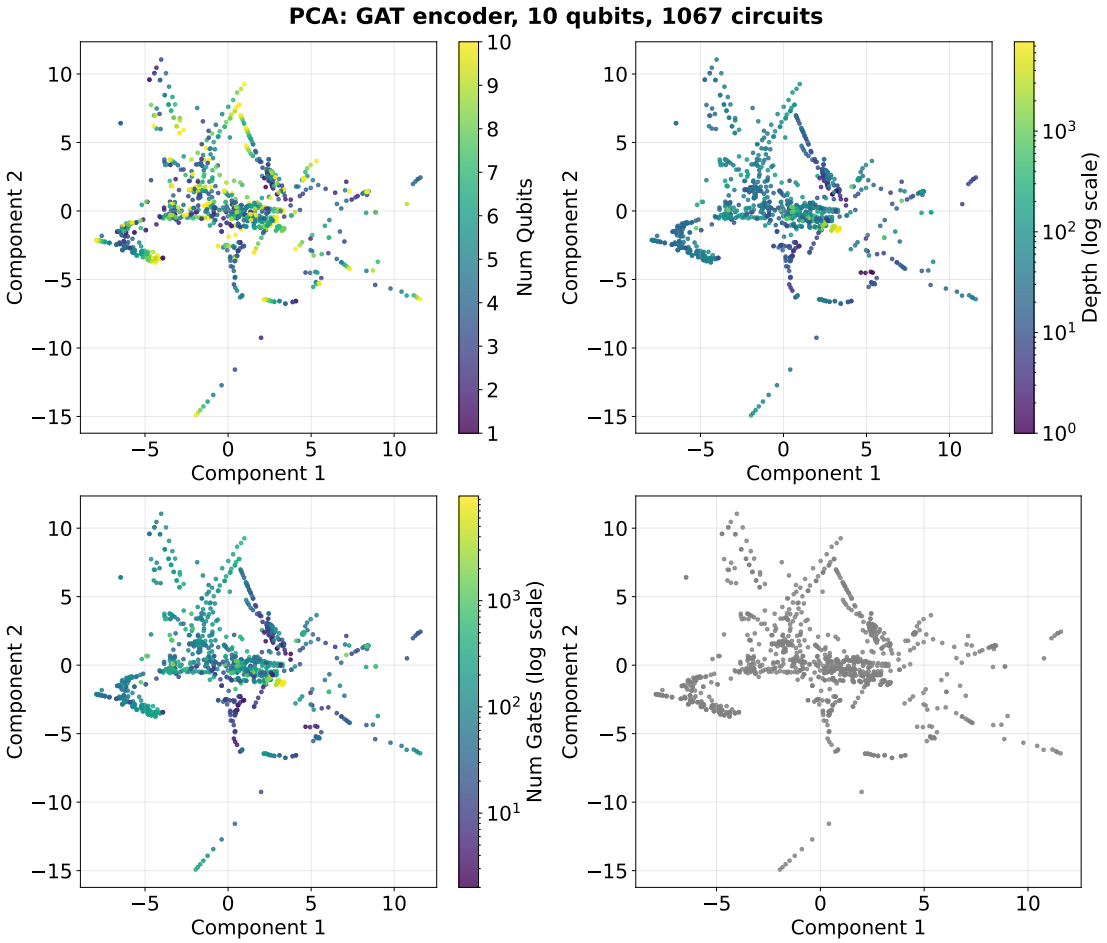


Figure 9.9: Two-dimensional PCA of the latent space generated by the GAT network. The embeddings represent a dataset comprising 1067 distinct quantum circuits, constrained to a maximum quantum register dimensionality of 10 qubits, extracted at training epoch 45.

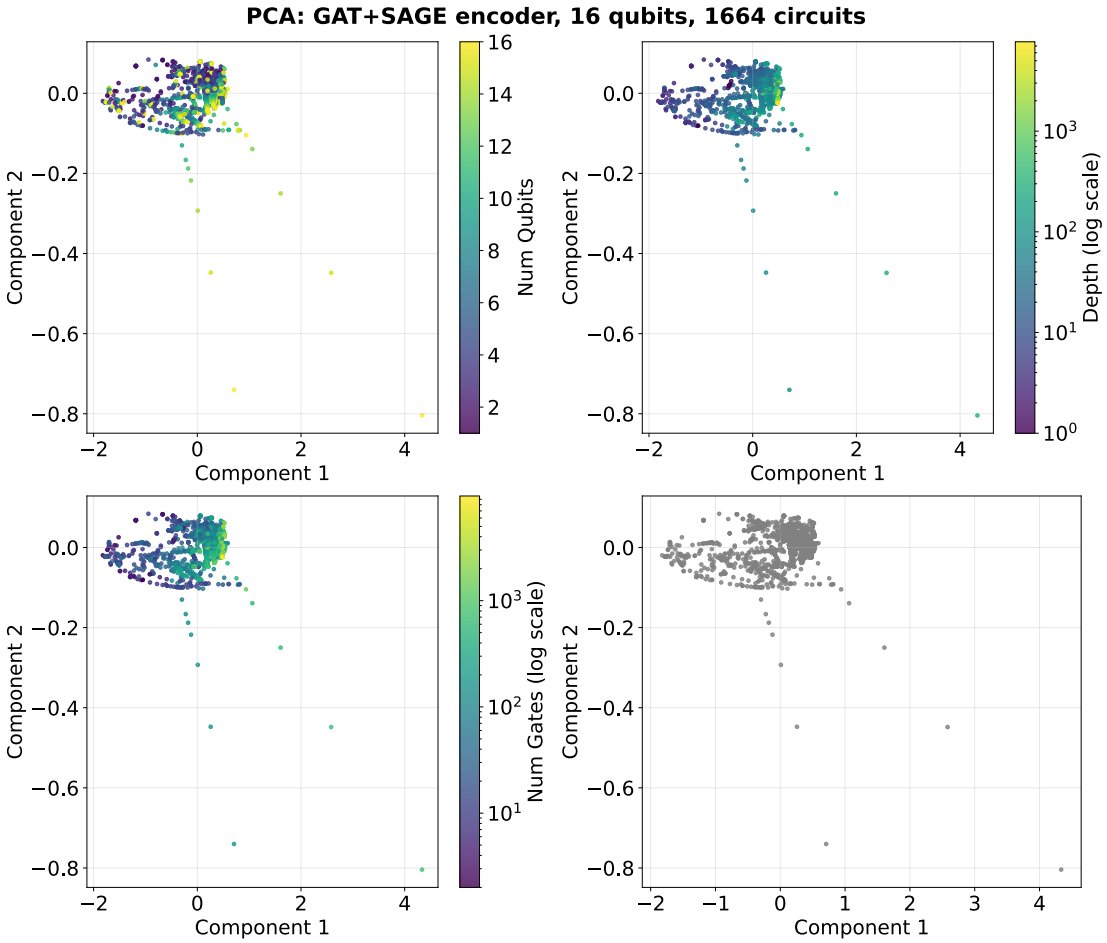


Figure 9.10: Two-dimensional PCA of the latent space generated by the mixed GAT+SAGE network. The embeddings represent a dataset comprising 1664 distinct quantum circuits, constrained to a maximum quantum register dimensionality of 16 qubits, extracted at training epoch 1.

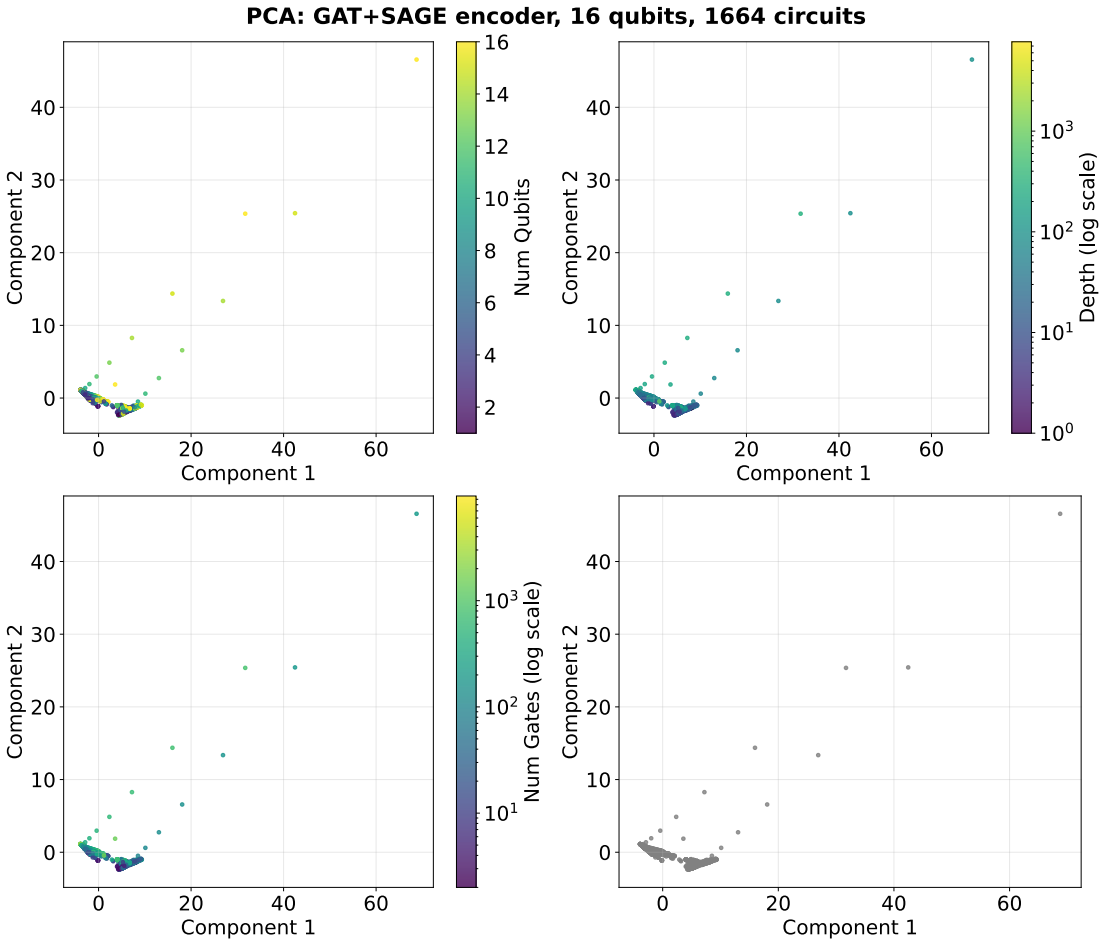


Figure 9.11: Two-dimensional PCA of the latent space generated by the mixed GAT+SAGE network. The embeddings represent a dataset comprising 1664 distinct quantum circuits, constrained to a maximum quantum register dimensionality of 16 qubits, extracted at training epoch 2.

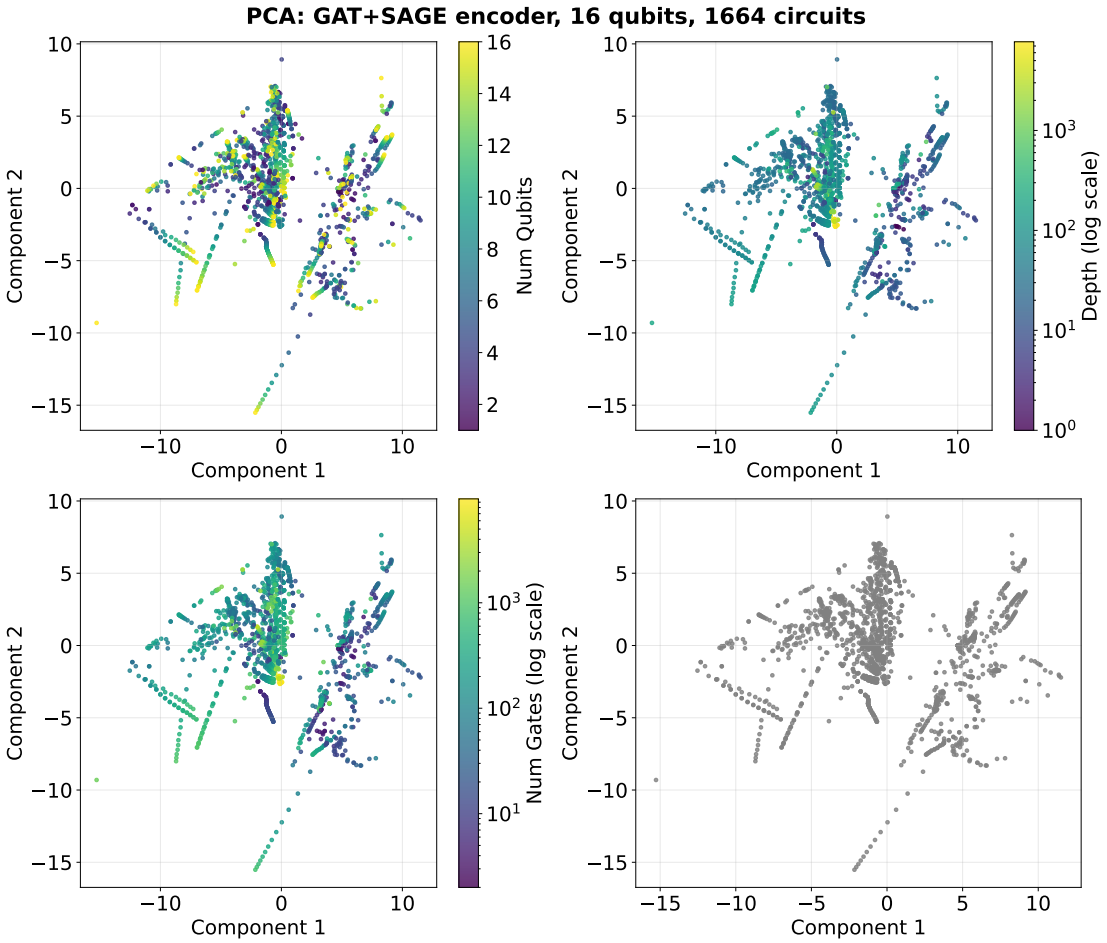


Figure 9.12: Two-dimensional PCA of the latent space generated by the mixed GAT+SAGE network. The embeddings represent a dataset comprising 1664 distinct quantum circuits, constrained to a maximum quantum register dimensionality of 16 qubits, extracted at training epoch 46.

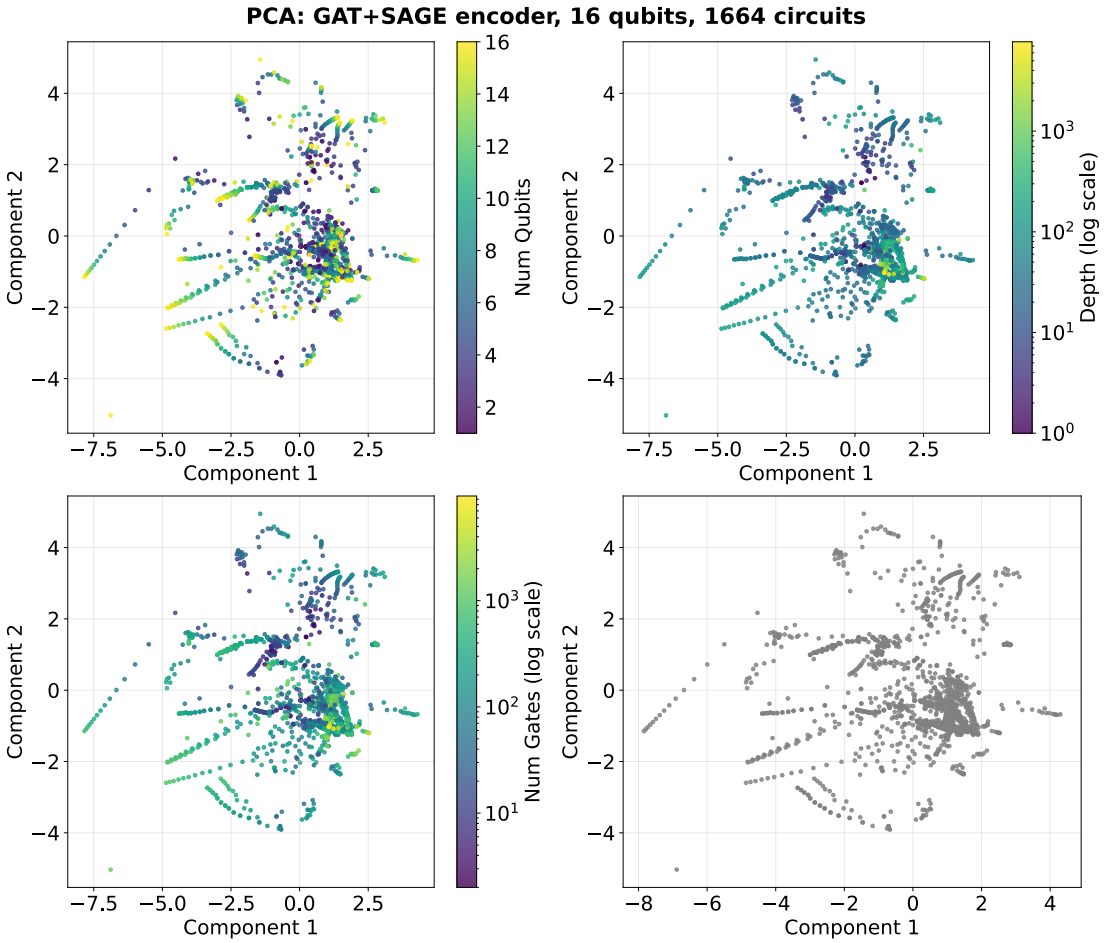


Figure 9.13: Two-dimensional PCA of the latent space generated by the mixed GAT+SAGE network. The embeddings represent a dataset comprising 1664 distinct quantum circuits, constrained to a maximum quantum register dimensionality of 16 qubits, extracted at training epoch 143.

Chapter 10

Summary and Experimental Results

The comprehensive evaluation of the framework across multiple quantum circuit datasets yields a diagnostic profile on the capabilities and fundamental limitations of the DAG-VAE architecture. The trials repeatedly reported the stagnation of the Total Loss function at a threshold greater than zero, highlighting structural limitations in the current architecture's capacity to perfectly learn and generate exact quantum circuits. The empirical data establish that the highest optimization yield achieved corresponds to an average of 80% reconstruction fidelity, associated with a maximum of 5% reduction in overall circuit depth, results that do not justify the choice of using this approach against traditional compilers.

The model demonstrates a moderate ability to encode continuous and macroscopic properties, while systematically failing to resolve discrete topological constraints. The training curves indicate that the Parameter Loss rapidly converges to values approaching zero, suggesting the architecture suitability for learning continuous variables. This observation is geometrically confirmed by the PCA projections. Across the 5, 8, and 10 qubit experiments, the projections exhibit well-defined gradients when mapped against the circuit's total depth or gate count. Conversely, the discrete nature of quantum operators imposes a bottleneck in discrimination. The training data explicitly show that the Qubit Loss systematically fails to drop below the 1.0 threshold across all dataset configurations, an issue confirmed by the PCA projections: circuits with different quantum register dimensionalities are overlapped in the latent space. This shows that the network lacks the capabilities required to separate and accurately reconstruct discrete graph connectivity, fundamentally limiting exact circuit synthesis.

Part IV

Conclusions and Future Perspective

The transition from theoretical quantum algorithms to executable hardware instructions is strictly bound by the physical limitations of NISQ computers. High error rates and short decoherence times impose compilation and optimization techniques to ensure the probability of successful computation. To offer an alternative to traditional rule-based and brute-force transpilation methodologies, this thesis introduced PANZEROTTO, a framework that proposes quantum circuit optimization as a GenAI task. Employing a DAG-VAE, the objective was to map the discrete structures of quantum circuits into a continuous latent space, allowing stochastic sampling of optimized circuits with reduced depth and gate counts accepting a trade-off between optimization and reconstruction fidelity.

Despite the theoretical validity of generative graph models in other domains, the results of this thesis highlight fundamental architectural limitations when applied to quantum circuit synthesis. Validation on different datasets (ranging from 5 to 16 qubits) demonstrated that the Total Loss function did not drop below a specific convergence threshold. The highest optimization yield achieved by the model was an average of 80% reconstruction fidelity corresponding to only a 5% reduction in the overall depth of the circuit. These results indicate that in its current formulation, the generative approach proposed in this thesis does not provide a competitive advantage over well-known deterministic compilers.

While the current architecture of the PANZEROTTO framework lacks the capabilities required for exact circuit synthesis, this thesis explores an innovative approach leveraging GenAI for Quantum Circuit Compilation paving the way for future research trajectories aimed at solving the current bottlenecks:

- Enrichment of Feature Extraction and Graph Representation: The current feature extraction pipeline relies on localized DAG properties. A possible improvement could be an enrichment of the quality of structural information provided to the model.
- Full Reinforcement Learning Integration: The hybrid "First learn, then optimize" training strategy struggled because the supervised VAE stage could not achieve a reliable reconstruction result. A promising future step is to overcome the reconstruction bottleneck and test a full training procedure.
- Refinement of the Wire Pooling Mechanism: While Wire Pooling was introduced to create persistent embeddings for specific qubits, the persistent Qubit Loss indicates it is insufficient for complex qubit routing.
- Hardware-Aware compilation: leveraging reward shaping techniques and Reinforcement Learning to create a simultaneous Hardware-Aware and Hardware-Unaware compiler.

In conclusion, although the PANZEROTTO DAG-VAE model does not yet offer significant advantages over traditional transpilation methodologies, this thesis outlined the challenges of treating quantum circuits as generative graphs.

Appendix A

Machine Learning

A.1 Altgraph

A.1.1 Gates

- single qubit gates: *x, y, z, h, s, t, id, sxdg, sdg, sx, tdg*[5]
- two qubit gates: *cx, cy, cz, swap, dcx, iswap, csdg, ecr, ch, cs, csx*[5]

A.1.2 Results

Table A.1: Average gate reduction per model[5]

Model	**Average Gates Reduced**	**Average Reduction%**
AltGraph GRU	35.67	37.55
AltGraph GCN	33.78	37.17
AltGraph DeepGMG	32.87	34.73
Qiskit	26.15	18.65
t$\|ket\rangle$	73.68	33.5
Quartz	–	30.1
Quarl	–	36.6
Monte Carlo Tree	–	30.0

Table A.2: Average depth reduction and test circuit MSE per model[5]

Model	**Average Depth Reduction%**	**Average MSE**
AltGraph GRU	37.75	.0074
AltGraph GCN	35.54	.27
AltGraph DeepGMG	40.08	.13

Appendix B

Panzerotto

B.1 Gate set

- ccx
- ch
- cp
- crx
- cry
- crz
- cswap
- cu
- cx
- cy
- cz
- h
- id
- p
- rx
- ry
- rz

- s
- sdg
- swap
- sx
- t
- tdg
- u
- u1
- u2
- u3
- x
- y
- z

Appendix C

Code repository

The software architecture is highly modular, executing a fully automated pipeline that embeds data preprocessing, neural network training and final compilation validation.

C.1 Workflow Orchestration and Initialization

main.py: This script is the central architectural entry point for the compiler. Scans JSON configuration files and orchestrates the sequential execution of the workflow phases (preprocessing, training, fine-tuning and compilation).

C.2 Model Architecture

src/model/PANZEROTTO.py: The principal PyTorch module coordinating the data flow between the Encoder,Decoder and Loss Function. It serves as a protection layer for masking tensors to prevent the prediction of invalid or unsupported quantum gates.

src/model/encoder.py: Implements a Graph Neural Network (GNN). The encoder dynamically supports architectures such as GraphSAGE or GAT, processing both node features and edge attributes. It employs a specific edge MLP to compute connection weights and utilizes wire pooling operations to aggregate node embeddings into distinct vectorial representations for each qubit. Finally, it performs a global graph embedding into the latent space.

src/model/decoder.py: Autoregressive decoder with GRU. The decoder sequentially synthesizes the optimized quantum circuit sampling each node based on its predecessors. It uses three independent Pointer Network attention heads to sample the target qubit for the current gate, and utilizes a continuous Gaussian policy to predict the exact rotational parameters. It also integrates a dedicated head for

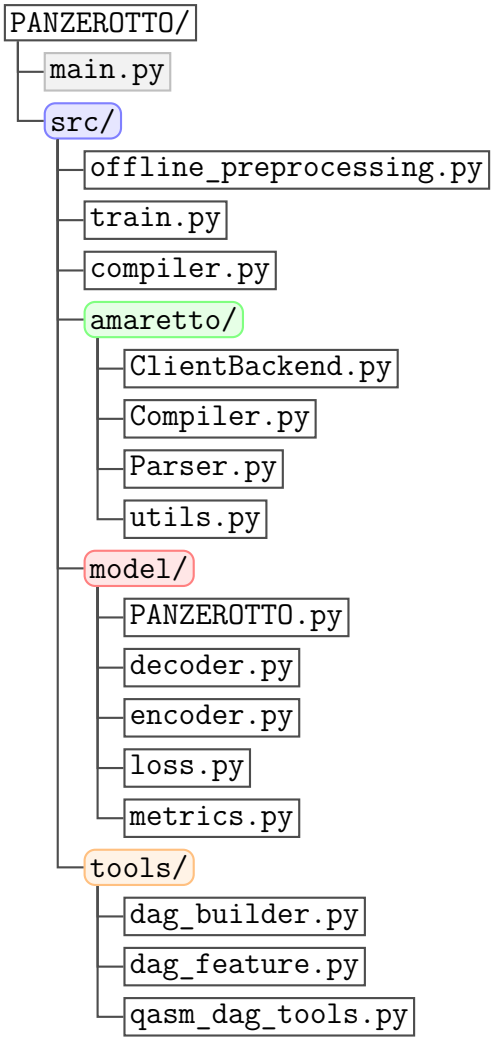

PANZEROTTO/
main.py
src/
offline_preprocessing.py
train.py
compiler.py
amaretto/
ClientBackend.py
Compiler.py
Parser.py
utils.py
model/
PANZEROTTO.py
decoder.py
encoder.py
loss.py
metrics.py
tools/
dag_builder.py
dag_feature.py
qasm_dag_tools.py

Figure C.1

End-Of-Sequence (EOS) prediction to dynamically control the generated circuit depth.

src/model/loss.py & src/model/metrics.py: The loss function sums the KL divergence, the cross-entropy for discrete gate and wire predictions, and the loss for continuous parameter. The metrics module calculates the Hellinger fidelity and the other metrics required for optimization.

C.3 Phase 1: Topological Abstraction and Offline Preprocessing

src/offline_preprocessing.py: This module manages the gathering and transformation of quantum circuits before the training loop. It loads OpenQASM 2.0 files, filters out circuits that exceed the target backend's qubit capacity or the maximum allowed gate operation count (e.g., >10,000 gates), and computes the ideal reference statevector using simulators such as Qiskit, Aer, or the Amaretto backend.

src/tools/qasm_dag_tools.py: This script converts Quantum Circuits DAG.

src/tools / dag_builder.py: This module creates the features vectors as in subsection 5.1.2.

C.4 Phase 2: Neural Training and Reinforcement Learning Fine-Tuning

src/train.py: Training loop. It supports multi-GPU distributed execution using PyTorch Distributed Data Parallel (DDP). The training procedure incorporates a distinct "two-stage" loop: an initial phase with heavy penalization of invalid qubit assignments, followed by a transition to a balanced VAE objective. In the fine-tuning phase, the script shifts to a Reinforcement Learning paradigm, employing an exponential moving average (EMA) baseline to calculate advantage values and optimize the policy via negative log-likelihood scaling.

C.5 Phase 3: Compilation Inference and Validation

src/compiler.py: Inference and validation of the trained model. It tests the trained model over the test dataset, generating multiple candidates and selecting the one yielding the highest reward.

Bibliography

[1] John Preskill. Quantum computing in the nisq era and beyond. *Quantum*, 2:79, August 2018.

[2] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[3] IBM Quantum. Ibm quantum platform computers, 2026. Accessed: 05/02/2026.

[4] Ge Yan, Wenjie Wu, Yuheng Chen, Kaisen Pan, Xudong Lu, Zixiang Zhou, Yuhan Wang, Ruocheng Wang, and Junchi Yan. Quantum circuit synthesis and compilation optimization: Overview and prospects, 2025.

[5] Collin Beaudoin, Koustubh Phalak, and Swaroop Ghosh. Altgraph: Redesigning quantum circuits using generative graph models for efficient optimization, 2024.

[6] Arianne Meijer - van de Griend. A comparison of quantum compilers using a dag-based or phase polynomial-based intermediate representation. *Journal of Systems and Software*, 221:112224, 2025.

[7] Igor L. Markov and Yaoyun Shi. Simulating quantum computation by contracting tensor networks. *SIAM Journal on Computing*, 38(3):963–981, January 2008.

[8] Aleks Kissinger and John van de Wetering. Reducing the number of non-clifford gates in quantum circuits. *Phys. Rev. A*, 102:022406, Aug 2020.

[9] Raban Iten, Romain Moyard, Tony Metger, David Sutter, and Stefan Woerner. Exact and practical pattern matching for quantum circuit optimization. *ACM Transactions on Quantum Computing*, 3(1), January 2022.

[10] Krishnageetha Karuppasamy, Varun Puram, Stevens Johnson, and Johnson P. Thomas. A comprehensive review of quantum circuit optimization: Current trends and future directions. *Quantum Reports*, 7(1), 2025.

[11] Aleks Kissinger and John van de Wetering. Pyzx: Large scale automated diagrammatic reasoning. *Electronic Proceedings in Theoretical Computer Science*, 318:229–241, May 2020.

[12] Alwin Zulehner and Robert Wille. Advanced simulation of quantum computations, 2018.

[13] Ali Javadi-Abhari, Matthew Treinish, Kevin Krsulich, Christopher J. Wood, Jake Lishman, Julien Gacon, Simon Martiel, Paul D. Nation, Lev S. Bishop, Andrew W. Cross, Blake R. Johnson, and Jay M. Gambetta. Quantum computing with Qiskit, 2024.

[14] Seyon Sivarajah, Silas Dilkes, Alexander Cowtan, Will Simmons, Alec Edgington, and Ross Duncan. t|ket⟩: a retargetable compiler for nisq devices. *Quantum Science and Technology*, 6(1):014003, November 2020.

[15] Manfredi Avitabile. Proposal for a multi-technology, template-based quantum circuits compilation toolchain. Master's thesis, Politecnico di Torino, 2021.

[16] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[17] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs, 2018.

[18] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018.

[19] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics.

[20] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[21] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2022.

[22] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning internal representations by error propagation*, page 318–362. MIT Press, Cambridge, MA, USA, 1986.

[23] Muhan Zhang, Shali Jiang, Zhicheng Cui, Roman Garnett, and Yixin Chen. D-vae: A variational autoencoder for directed acyclic graphs, 2019.

[24] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[25] Andrew W. Cross, Lev S. Bishop, John A. Smolin, and Jay M. Gambetta. Open quantum assembly language, 2017.

[26] Antonio Tudisco, Deborah Volpe, Giacomo Orlandi, and Giovanna Turvani. Graph neural network-based predictor for optimal quantum hardware selection, 2025.

[27] Ville Bergholm, Josh Izaac, Maria Schuld, Christian Gogolin, Shahnawaz Ahmed, Vishnu Ajith, M. Sohaib Alam, Guillermo Alonso-Linaje, B. Akash-Narayanan, Ali Asadi, Juan Miguel Arrazola, Utkarsh Azad, Sam Banning, Carsten Blank, Thomas R Bromley, Benjamin A. Cordier, Jack Ceroni, Alain Delgado, Olivia Di Matteo, Amintor Dusko, Tanya Garg, Diego Guala, Anthony Hayes, Ryan Hill, Aroosa Ijaz, Theodor Isacsson, David Ittah, Soran Jahangiri, Prateek Jain, Edward Jiang, Ankit Khandelwal, Korbinian Kottmann, Robert A. Lang, Christina Lee, Thomas Loke, Angus Lowe, Keri McKiernan, Johannes Jakob Meyer, J. A. Montañez-Barrera, Romain Moyard, Zeyue Niu, Lee James O'Riordan, Steven Oud, Ashish Panigrahi, Chae-Yeun Park, Daniel Polatajko, Nicolás Quesada, Chase Roberts, Nahum Sá, Isidor Schoch, Borun Shi, Shuli Shu, Sukin Sim, Arshpreet Singh, Ingrid Strandberg, Jay Soni, Antal Száva, Slimane Thabet, Rodrigo A. Vargas-Hernández, Trevor Vincent, Nicola Vitucci, Maurice Weber, David Wierichs, Roeland Wiersema, Moritz Willmann, Vincent Wong, Shaoming Zhang, and Nathan Killoran. Pennylane: Automatic differentiation of hybrid quantum-classical computations, 2022.

[28] Robert Wille, Lucas Berent, Tobias Forster, Jagatheesan Kunasaikaran, Kevin Mato, Tom Peham, Nils Quetschlich, Damian Rovara, Aaron Sander, Ludwig Schmid, Daniel Schoenberger, Yannick Stade, and Lukas Burgholzer. The MQT handbook: A summary of design automation tools and software for quantum computing. In *IEEE International Conference on Quantum Software (QSW)*.

[29] Ang Li, Samuel Stein, Sriram Krishnamoorthy, and James Ang. Qasmbench: A low-level qasm benchmark suite for nisq evaluation and simulation. *arXiv preprint arXiv:2005.13018*, 2021.

[30] Ang Li, Samuel Stein, Sriram Krishnamoorthy, and James Ang. Qasmbench: A low-level quantum benchmark suite for nisq evaluation and simulation. *ACM Transactions on Quantum Computing*, 2022.

[31] https://github.com/veri-q/benchmark.

[32] *Legion Cluster, HPC@POLITO*.

[33] *Distributed Data Parallel, PyTorch*.

[34] Christian Conti, Deborah Volpe, Mariagrazia Graziano, Maurizio Zamboni, and Giovanna Turvani. Amaretto: Enabling efficient quantum algorithm emulation on low-tier fpgas. In *2024 31st IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, page 1–4. IEEE, November 2024.