TOWARDS SCALABLE CIRCUIT PARTITIONING IN MULTI-CORE QUANTUM ARCHITECTURES WITH GRAPH REINFORCEMENT LEARNING

ARNAU ESTEBAN MÁRQUEZ

Thesis supervisor
SERGI ABADAL CAVALLÉ (Department of Computer Architecture)

Thesis co-supervisor
PAU ESCOFET I MAJORAL

Degree
Bachelor's Degree in Informatics Engineering (Computing)

Bachelor's thesis

Facultat d'Informàtica de Barcelona (FIB)

Universitat Politècnica de Catalunya (UPC) - BarcelonaTech

23/10/2024

Abstract

Quantum computers present an undeniable potential to solve classically intractable problems. However, the scalability of quantum architectures remains a principal challenge. A promising solution is the use of multi-core quantum architectures, which have to deal with new hardware and algorithmic challenges. The problem addressed in this work is partitioning quantum circuits among cores, which is an NP-hard problem. This thesis presents a novel approach for scalable circuit partitioning in multi-core quantum architectures using deep reinforcement learning (DRL) with graph neural networks (GNN), which allows training circuits of different number of qubits, unlike previous work that is typically constrained to a fixed number. The findings highlight a starting point to the potential of combining DRL and GNNs for solving more complex quantum problems.

Keywords: Scalable circuit partitioning, multi-core quantum architectures, deep reinforcement learning, graph neural networks

Acknowledgements

I would like to express my sincere gratitude to my thesis supervisor, Sergi Abadal, for giving me the opportunity to conduct this research and for his invaluable guidance throughout the project. His support and encouragement have been fundamental in shaping this work.

I am also deeply thankful to my co-supervisor, Pau Escofet i Majoral, for his continuous key insights and feedback during the research. His expertise, thoughtful advice and availability at all times have greatly contributed to the successful completion of this thesis.

The invaluable insights provided by my former instructor in artificial intelligence, Víctor Giménez, not only clarified complex topics but also significantly enriched my understanding of the field.

Finally, I would like to extend my appreciation to all the teachers and staff at FIB for providing me with an enriching academic environment throughout the entirety of my Computer Science Bachelor, which has equipped me with the knowledge and skills necessary to complete this work and all the upcoming challenges.

Contents

Chapter 1

Introduction

1.1 Overview

Based on classical physics and boolean algebra, classical computers undeniably reign over today's computing context, being the pillar of the digital revolution and driving outstanding advances throughout the years. Nonetheless, some problems are classically intractable due to their nature, and it is at this point where quantum computers gain ground. Particularly, quantum computers could potentially solve intractable problems, like RSA encryption [1], which is one of the main promises, thanks to their ability to perform exponentially faster than classical computers for a particular set of algorithms. This is possible due to their capability to perform parallel computations through quantum principles like superposition and entanglement [2]. In fact, the high potential of this field has encouraged important firms like IBM to hugely invest in this field [3], which has become nowadays a competitive race area.

The basic unit of classical computers is the bit, which can only have two states (0 or 1). For quantum computers it is the qubit, whose state is a linear combination of two basis states. This property of superposition of the two states, among others, justifies the exponential benefits of this technology, but at the same time it opens the door to various challenges when developing quantum computers.

Decoherence is one of the main causes that challenges scaling up the number of qubits in a quantum processor. It refers to the loss of quantum information due to interactions with the environment. While, to date, IBM has produced a 1121 qubit quantum processor [4], which is an outstanding milestone, quantum decoherence remains the primary obstacle to scaling quantum computers since it increases the number of physical qubits that are necessary to implement a single logical qubit. For instance, some research estimate that around 20 million of noisy qubits are needed to break the 2048-bit RSA encryption [5].

1.2 Motivation

Given the current technological advancements, an approach that is considered to be more feasible to scale up the number of qubits is the use of multi-core quantum architectures, where qubits are distributed in separate chips (cores) that need to be communicated with quantum-coherent links.

During the execution of a quantum program, the qubits interact through quantum gates. Nevertheless, due to the hardware limitations, in-cluster communications (within the same core) are much faster and efficient than non-local communications (between different cores) [6]. Thus, interacting qubits must be in the same core. That is why, during the execution of a program, they move across different cores to respect the operations dependencies.

Figure 1.1 illustrates the architecture of a multi-core quantum computer.

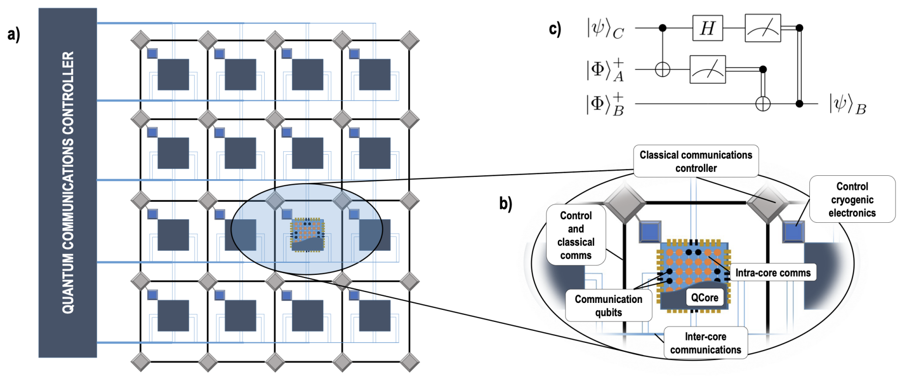

a)
QUANTUM COMMUNICATIONS CONTROLLER
c)
$|\psi\rangle_C$
$|\Phi\rangle_A^+$
$|\Phi\rangle_B^+$
H
$|\psi\rangle_B$
b)
Classical communications controller
Control cryogenic electronics
Control and classical comms
Intra-core comms
Communication qubits
QCore
Inter-core communications

Figure 1.1: Multi- chip quantum computer full view. (*a*) 2D diagram of a multi- chip architecture. (*b*) Enumeration of the components, including intra- and inter-core communications. (*c*) Circuit for quantum teleportation (qubit inter-core movement) *Source:* [7]

The interactions between qubits are known at compile time; therefore, it is the compiler's job to map virtual qubits into cores in order to reduce the number of non-local communications. Despite this, research on this aspect remains scarce.

State-of-the-art algorithms for qubit mapping, such as FGP-rOEE [6], struggle to scale up the number of qubits in a reasonable execution time. Thus, other alternatives employ artificial intelligence to address the same issue [8, 9]. However, they are not exempt from significant limitations, e.g., limited real-world datasets and variations in the number of qubits the AI models are trained with.

To date, although other approaches have shown promise, they still face significant challenges, particularly in terms of efficiency and generalization across different quantum computing architectures. These limitations highlight the need for further advancements in the field. The ultimate goal is to develop a truly universal quantum compiler capable of handling any quantum application, regardless of the number of qubits or cores.

While this thesis does not achieve the ultimate goal, it is expected to serve as a starting point on the path toward the development of a universal quantum compiler. By addressing the challenge of designing a solution capable of handling varying numbers of qubits, this study lays the groundwork for future research aimed at reaching this ambitious objective.

1.3 Problem definition

The problem arises partitioning quantum circuits in multi-core quantum architectures. In these architectures, qubits within the same core exhibit a high connectivity, meaning that the application of quantum gates among them can be performed with minimal latency. Therefore, whenever qubits need to interact, it is essential that they are allocated within the same core. However, since the number of qubits per core has a maximum capacity and must remain fixed, during the execution of a quantum algorithm, virtual qubits need be transferred across cores to perform gate interactions with other qubits. This is challenging since quantum qubit inter-core movements are very costly; therefore, it is crucial to minimize the number of quantum state transfers across cores. This is the global objective of qubit mapping.

Efficient circuit partitioning is critical for the overall performance of quantum algorithms, particularly as quantum architectures scale. Poor partitioning can lead to excessive inter-core communication, which introduces latency and can diminish the advantages of quantum computing. By solving the circuit partitioning problem effectively, it is ensured that multi-core quantum architectures remain a viable path for scaling quantum computers to handle more complex, real-world problems.

1.4 Objectives

The main objective of this research is to develop a deep reinforcement learning agent with graph neural networks (Graph Reinforcement Learning) in the context of quantum computing, more specifically for qubit mapping in quantum multi-core architectures.

This main goal is divided into the following sub-objectives.

- Deeply investigate the state-of-the-art partitioning techniques for quantum circuits.
- Create a balanced dataset with random and synthetic circuits.
- Create an AI agent that can be trained with circuits of different number of qubits.
- Create an efficient agent with a feasible training time.
- Demonstrate that the agent effectively learns to map qubits to cores.

1.5 Modification of objectives

Some objectives have been modified after deeply analyzing the problem.

While, at the beginning, one of the main objectives was to outperform the state-of-the-art algorithms, it has been concluded that this purpose was excessively ambitious. At this stage, it is hardly feasible to outperform the greedy state-of-the-art algorithms using AI, since significant development, study and work remains to be done in this area. Besides, the short timeline of this project is another downside to reach this goal.

That is why, this specific objective is adapted to creating the DRL agent and demonstrate that it is capable to learn, providing a promising starting point for further research. The rest of the objectives remain unchanged.

1.6 Context at FIB

This research is carried out as a thesis for the Bachelor's Degree in Informatics Engineering at Polytechnic University of Catalonia. It belongs to the major of Computing, more specifically to the field of Artificial Intelligence.

It is done within the framework of the investigation groups *Barcelona Neural Networking Center (BNN)* and *NaNoNetworking Center in Catalonia (N3Cat)*.

- *BNN*'s mission is carrying out fundamental research in the field of graph neural networks applied to computer networks, which are used throughout this research.
- *N3Cat*'s mission is carrying out fundamental research on nanonetworks, which are used in various quantum computer applications.

1.7 Stakeholders

A successful result of this thesis would benefit a wide number of actors.

In the first place, quantum computing is in its very early stages, there is still a long way to go in this field. Due to its current state and its high potential, a large number of universities and research institutions are diving deep into this industry. This project is therefore directed towards quantum researchers and developers, who could make use of the results of this thesis to continue and improve their research and projects. Companies specializing in quantum computing, particularly in software aspects and development of compilers, like ParityQC, Horizon Quantum, Terra Quantum, are directly interested in any advance in this field. Furthermore, quantum computing hardware manufacturers such as Google, IBM, Microsoft and Intel, among others, are directly interested in any advance in this field.

Going further, the customers of these enterprises that require the computational power of these clusters may be also indirect beneficiaries of this research in the long term.

On balance, any actor who researches, develops, sells or uses quantum computing systems, directly or indirectly, is stakeholder of this research.

1.8 Setbacks and Risks

This research may suffer from important setbacks and risks that need to be analyzed in advance.

First of all, the use of graph reinforcement learning may lead to long execution times during training. It is crucial to set an efficient approach for the DRL agent so that the balance between training time and learning is appropriate. The training time cannot be substantially high as it is important to contrast and evaluate different configurations. Hence, if the training process takes too long, severe difficulties will be found when evaluating the model. That is why it is necessary to use a high-performing hardware.

Additionally, it is not guaranteed that the model will learn due to the complexity of the problem. There is a large amount of different hyperparameters, not only for the agent, but also for the GNN, which make the problem even more complex. Thus, different configurations will need to be evaluated in order to reach the most optimal one, owing to the fact that it is not assured that the model will achieve the goals of the project (see Section 1.4).

In addition, due to the computational complexity, even if a very efficient model is developed for a certain number of training iterations, perhaps this number is not enough and the model needs a much larger number of iterations to learn.

As aforementioned in Section 1.6, this work is conducted with the feedback and support of N3Cat and BNN groups, whose experts provide significant insights into the relevant fields of this research.

Chapter 2

Background

In this section, key concepts necessary to understand this thesis are described.

2.1 Quantum computing: Fundamental concepts

As introduced in Section 1.1, qubits constitute the basic unit of quantum information, whose state is described by a linear superposition of the two basis states $|0\rangle$ and $|1\rangle$. Operations on qubits are performed through quantum gates, which can be classified as one-qubit or multi-qubit gates.

Quantum algorithms are expressed as quantum circuits, which primarily consist of a sequence of quantum gates on qubits. Since in most quantum hardware only single-qubit and two-qubit gates can be implemented, it is necessary a *gate decomposition* mechanism that breaks down multi-qubit gates into single-qubit and two-qubit gates. Figure 2.1 illustrates an example of a 5-qubit circuit.

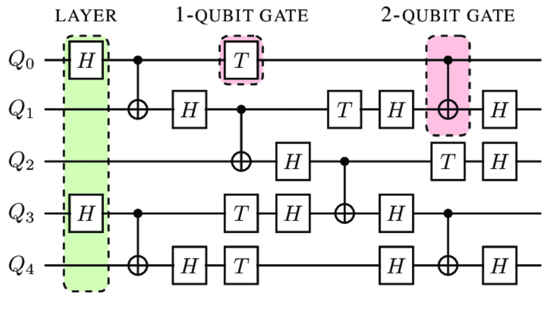


Figure 2.1: Example of a 5-qubit quantum circuit from [10], with each horizontal line representing the time-evolution of the state of a single virtual qubit. *Source:* [11]

As advanced in Section 1.2, currently, multi-core quantum architectures are envisioned to scale up the number of qubits in a circuit. However, this comes with a physical limitation: whenever two qubits need to interact (i.e., there is a two qubit gate involving those qubits) they need to be located in the same core. Therefore, the virtual qubits will inevitably need to be transfered to different cores throughout the execution of a program in order to align to the hardware limitations. The assignation of virtual qubits to physical qubits in the hardware (in cores in the multi-core scenario) is named **qubit mapping**.

Single-core quantum architectures are constructed with specific topologies, which limit the connectivity between qubits. Figure 2.2 depicts the qubit connectivity of the IBM's quantum processor *ibmq_dublin*. Therefore, according to this structure, two-qubit gates can only be applied on adjacent qubits (nodes of the graph).

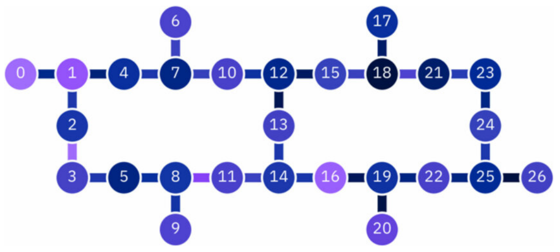


Figure 2.2: Qubit connectivity of ibmq_dublin. *Source:* [12]

To this purpose, SWAP gates are usually introduced to the circuit to exchange the states of two physical qubits. When the connectivity between qubits is insufficient for the required operations (quantum gates), SWAP gates are employed in order to rearrange the qubits accordingly. Nonetheless, adding SWAP gates to a circuit introduces additional overhead. For this reason, optimizing the qubit mapping while minimizing the number of SWAP gates is an essential task in quantum computing.

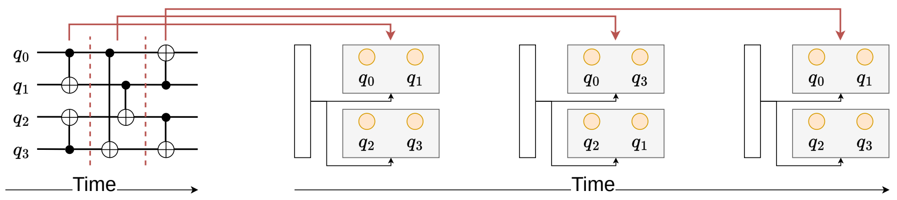


Figure 2.3: Multi-core qubit mapping. *Source: [13]*

This thesis focuses on multi-core quantum architectures. As detailed in Section 1.3, each core contains a fixed number of qubits with high internal connectivity, while non-local communications between cores incur high latency. This underscores the necessity of allocating interacting qubits within the same core.

As illustrated in Figure 2.3, in each time step, interacting qubits are allocated in the same core. During the execution time, qubits change of core to perform their interaction with other qubits.

2.2 Quantum circuits

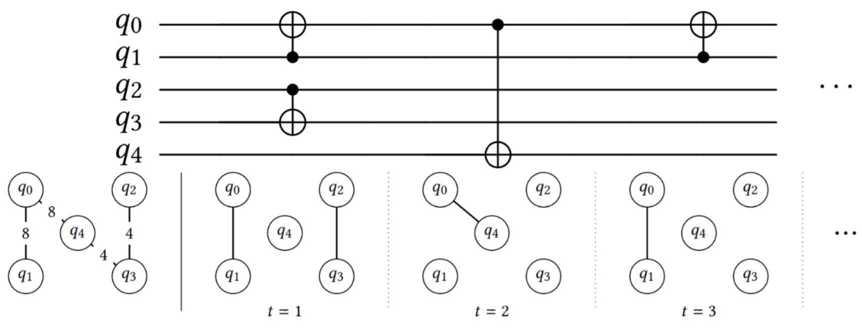


Figure 2.4: *(Top)* Interactions between qubits in a quantum program, read from left to right. *(Bottom)* Interaction graphs at each time slice. *Source:* [9]

Quantum circuits are basically step-by-step procedures in which qubits interact between them using quantum gates. Multi-qubit gates can be decomposed into a chain of single- and two-qubit gates. Since one-qubit gates do not need any special mapping to hardware, in this thesis, only two-qubit gates are considered. Some of these interactions may be produced in parallel, similarly to what happens with classical algorithms. Figure 2.4 shows a graph diagram that represents the interactions between qubits throughout the execution time of the program. Each step of parallel interactions is defined as time slice. For instance, in the diagram in Figure 2.4, the time slices are as follows:

- Time slice 1: interaction between the qubits *q0* and *q1*, interaction between the qubits *q2* and *q3*
- Time slice 2: interaction between the qubits *q0* and *q4*
- Time slice 3: interaction between the qubits *q0* and *q1*

The bottom of Figure 2.4 depicts a chain of interaction graphs, one for each time slice of the circuit. The leftmost graph depicts the aggregation of each interaction graph. Specifically, in this example case, the weights of the edges are proportional to the number of future interactions between the concerned qubits (represented as nodes). This is a possible way to represent a quantum program using a graph, which could be the input to a graph neural network (defined in Section 2.4).

2.3 Deep Reinforcement Learning

2.3.1 Deep Learning

Deep learning (DL) is a field of machine learning that creates models inspired by the structure and functionality of the human brain. They try to learn representations of hierarchical data through the use of neural networks, which are computational models composed of nodes, called artificial neurons, that are interconnected between them and, in the case of deep learning, they are organized in multiple non-linear layers.

2.3.2 Reinforcement Learning

Contrary to supervised learning, reinforcement learning (RL) is a branch of machine learning in which an agent learns to make decisions with no need of a labeled dataset, it learns by interacting with a changing environment.

RL's main components are the following:

- *Agent*: The decision-maker that interacts with the environment.
- *Environment*: An external system where the agent interacts (by taking actions), and gives feedback to the agent in the form of new states (*observations*) and cumulative rewards.
- *State*: State of the environment in a specific moment.
- *Action*: Based on the environment's current state observed by the agent, this one selects an action that affects the environment.
- *Reward*: Numerical value that indicates the immediate benefit or penalty as a result of the agent's taken action.

A RL agent learns by trying to maximize the total reward. It tries to find an optimal strategy where all its actions maximize the reward. If the reward is well connected to the objective of the system, then the agent will be able to solve the assigned problem.

Besides, it is also crucial to strike a balance between exploration and exploitation. Exploration involves trying out new actions to discover their potential rewards, while exploitation focuses on leveraging known actions that have previously yielded positive outcomes. Maintaining a small percentage of randomness in decision-making is essential (ε-greedy strategy), as it helps to avoid the risk of converging on a local optimum, which is a situation where the algorithm becomes stuck in a suboptimal solution since it continues to exploit familiar actions instead of exploring new possibilities.

2.3.3 Components of Deep Reinforcement Learning

Deep reinforcement learning (DRL) is a subfield of artificial intelligence and machine learning that combines the representation learning capabilities of deep learning with the making-decisions capabilities of reinforcement learning. DRL agents learn to make decisions while they interact with the environment.

DRL agents have mainly the following two components:

- *value function*: Function that estimates the cumulative reward of taking an action in a given environment state.
- *policy*: Strategy used by the agent to make decisions.

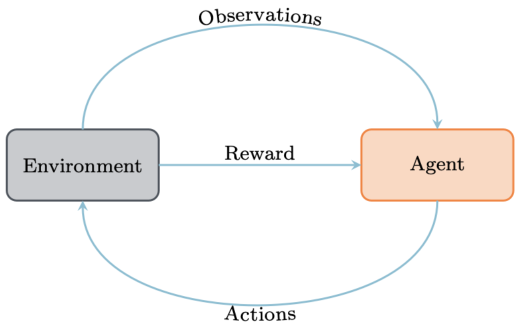


Figure 2.5: DRL agent and its environment. *Source:* [14]

In DRL, deep neural networks are used to approximate the value function, the policy, or both. Figure 2.5 depicts a global scheme of a RL architecture.

2.3.4 Q-learning

Q-learning [15] is a reinforcement learning algorithm that enables an agent to learn the value of an action taken in a particular state. This value is referred as the *Q-value*, which describes how effective or ineffective the taken action has been. Throughout its policy, it decides which action to take at each step.

Q refers to the learnable differentiable function that the algorithm learns and computes to calculate the expected value (or future reward) over the current and succeeding steps, based on a combination state-action:

$$Q : S \times A \to \mathbb{R} \tag{2.1}$$

where S is the set of states and A the set of actions.

Traditional Q-learning uses a table $|S| \times |A|$ to approximate the Q-function.

The fundamental update rule for the Q-value is the following:

$$Q(s,a) = Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right] \tag{2.2}$$

where:

- s is the state, and a is the action
- α is the learning rate
- r is the immediate reward after taking action a in state s
- γ is the discount factor, which is a number between 0 and 1, and determines the importance of future rewards
- s' is the next state reached after taking action a at state s

Through repeated interactions (steps) with the environment, the agent converges towards optimal Q-values, allowing it to select actions that maximize the cumulative reward.

2.3.5 Deep Q-learning

While traditional Q-learning uses a table as a Q-function, *deep Q-learning* (DQN) [16] is a variant that integrates deep learning techniques to approximate the Q-function. As the complexity of the problem increases, the complexity of the environment, the state and action spaces become too large to be represented in a Q-table. In these cases, DQN gains ground and employs deep neural networks to approximate the Q-function.

Similarly to the Q-learning update Equation (2.2), the loss function used to update the network's weights is usually computed using the mean squared error (MSE) between the predicted Q-values and the target Q-values.

$$L(\theta) = \mathbb{E}\left[(y - Q(s,a;\theta))^2 \right] \tag{2.3}$$

where:

- $y = r + \gamma \max_{a'} Q(s', a'; \theta^-)$ is the target Q-value
- θ represents the parameters of the **main** network

- θ^- represents the parameters of the **target** network

The target network is a critical component in a DQN agent to stabilize the training process. It is a separate neural network that is a delayed copy of the main network (Q-function). As observable in the loss function (Equation 2.3), the target network is used to calculate the target Q-values during training. By using a separate network for target values, it helps to reduce the correlation between the Q-value updates, leading to more stable learning. The target network is updated less frequently than the main network. Normally, its weights are copied from the main network at regular intervals rather than being updated in each training iteration. This slow update helps to prevent severe oscillations and divergence in the learning process.

2.4 Graph Neural Networks

Due to the nature of the quantum circuits, which can be easily represented as graphs, graph neural networks (GNN) are used throughout this thesis. They belong to a class of neural networks, whose architecture is of type *graph-in, graph-out*, which means that the input is a graph and the output is the same graph converted, which can be subsequently pooled. GNNs represent a great advantage over traditional neural networks when working with relational datasets, where the data can be naturally and compactly represented as a graph. Features can be represented in graph nodes, edges, or both. GNNs can adopt different architectures, presented in Figure 2.6, which offer advantages and disadvantages depending on the nature of the problem.

Generally, GNN are composed of the following components:

- *Node embeddings*: Features can be embedded into the graph nodes. During the forward pass, their state is updated based on the state of the same node and information from its neighbourhood.
- *Edge representations*: Some GNN architectures may also consider features for the edges, like weights or other characteristics about the relation between nodes.
- *Message passing*: In most cases, GNNs propagate information through the edges to their neighbourhood with a *message passing function.*
- *Update function*: During message passing, nodes update their own state through an *update function* that summarizes the information received by the adjacent nodes.
- *Readout function*: This function summarizes the overall state of the graph to obtain a graph-level embedding, which is the output of the GNN.

Depending on the chosen architecture, the components above are defined in one way or another.

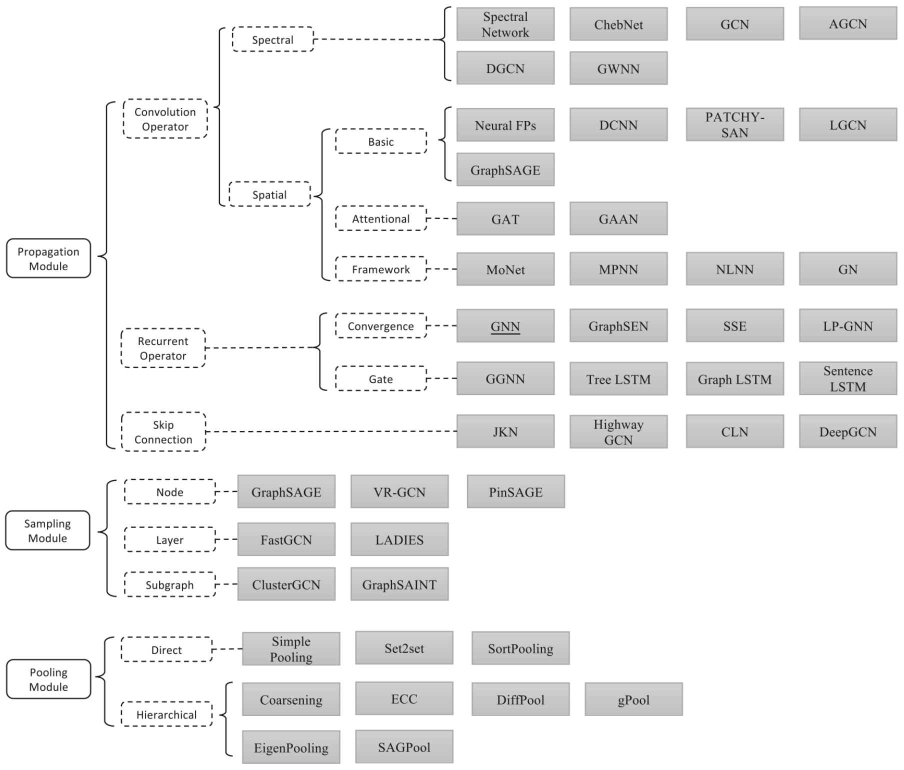


Figure 2.6: Architectures of graph neural networks. *Source:* [17]

2.4.1 Message Passing Neural Network

Following the successful approaches of the studies [18] and [9], in this thesis, message passing neural networks (MPNN) are employed. MPNN is a type of GNN, which can adopt different structures. However, in this section, a description is made focused on the model used in this research. In this case, it consists of an MPNN that operates on undirected graphs with both node and edge features.

The forward pass of the MPNN has two phases: a message passing phase and a readout phase.

Message passing phase
In this phase, for T time steps (a hyperparameter), the nodes update their hidden state h_v^t based on received messages from their adjacent nodes. These messages are created through a message function M_t, then they are aggregated, and an update function U_t updates the hidden state of the nodes.

The received messages by a node v are aggregated as follows:

$$m_v^{t+1} = \sum_{w \in N(v)} M_t(h_v^t, h_w^t, e_{vw}) \tag{2.4}$$

where $N(v)$ denotes the adjacent nodes of v, and e_{vw} are the edge features between the nodes v and w.

Thereafter, at each time step t, the hidden states of the nodes are updated.

$$h_v^{t+1} = U_t(h_v^t, m_v^{t+1}) \tag{2.5}$$

Both U_t and M_t are learnable differentiable functions (e.g., neural networks).

Readout phase

After T time steps, the readout phase pools the resulting graph using the final node hidden states (h_v^T) through the learnable differentiable Readout function R. The latter, depending on the task at hand, can be applied individually to each node, or to the whole graph (by previously aggregating the final hidden states of the nodes).

Figure 2.7 illustrates an example of MPNN. Depending on the application, the architecture of the MPNN could vary.

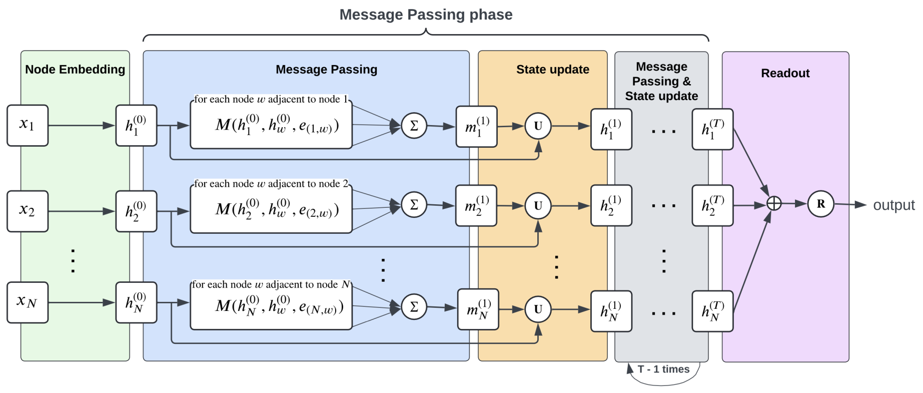


Figure 2.7: Flow diagram of an MPNN composed of three phases: initial node embedding, message passing and readout. x_i are the node features of a graph of N nodes, which are embedded to a first state $h_i^{(0)}$. A node hidden state at time step t is represented as $h_i^{(t)}$. Edge features are represented as $e_{(v,w)}$. Functions and operators are indicated as circles, where $\oplus$ means an aggregation (e.g., sum, mean, max), M is the message function, U the update function and R the readout function. For each node, its received messages are calculated with M based on the adjacent nodes and their edge features, and finally summed ($\sum$).

Chapter 3

State Of The Art

In this section, relevant alternatives for multi-core quantum compilation are analyzed. Firstly, the heuristic procedure FGP-rOEE [6] is described. Subsequently, alternatives that use artificial intelligence are compared. The limitations of each approach are discussed, followed by an explanation of how this thesis addresses and surpasses them.

3.1 FGP-rOEE

FGP-rOEE [6], which stands for *Fine Grained Partitioning using relaxed Overall Extreme Exchange*, is a state-of-the-art algorithm. Although, in some cases, other approaches have outperformed FGP-rOEE, this one is analyzed since it is considered the base of future related studies. Other approaches are the *Quadratic Unconstrained Binary Optimization* [19] and the *Hungarian Qubit Assignment* [20].

FGP-rOEE uses a relaxed version of the *Overall Extreme Exchange* (OEE) algorithm. OEE is a heuristic-based solution for the k-way graph partitioning problem [21], which consists of allocating graph nodes into k partitions equally. The procedure involves consecutively exchanging pairs of nodes to maximize the reduction of the cut between partitions, where the cut is defined as the sum of the edge weights crossing different partitions. The algorithm runs until there is no exchange that reduces the partitioning cost.

FGP-rOEE adapts the qubit mapping problem to a graph partitioning problem, where the nodes represent qubits, and the edge weights represent their gate interactions (as seen in Figure 2.4). In this case, a lookahead function is used to calculate the edge weights based on the immediate interactions and the ones from future time slices.

Since OEE addresses the graph partitioning problem, which is NP-hard, FGP-rOEE employs a relaxed version to improve its efficiency and performance. Instead of running the procedure until the optimal assignment is found, it runs until the partitions become valid, i.e., all interacting qubits (nodes) are in the same partitions (cores). With this modification, FGP-rOEE finds a favorable partition in cases where it would be unfeasible with OEE.

Limitations

Although FGP-rOEE has proven significant results for quantum circuit partitioning in multi-core architectures, it is subject of certain limitations. While real-world scenarios increasingly require more qubits and grow in complexity, FGP-rOEE may face scalability issues and struggle to partition large circuits within a reasonable execution time [13, 22].

3.2 Comparative Approaches

Other relevant studies employ artificial intelligence approaches to address the scalability issues of FGP-rOEE since trained AI models significantly improve execution time.

Ruizhe Yu Xia's master's thesis [9] aims to imitate FGP-rOEE algorithm's [6] results using graph neural networks (GNN), more specifically message passing neural networks (MPNN). Thanks to the lower computational complexity of GNNs, which is in its basic form $\mathcal{O}(n+m)$, Yu Xia's work promises a more efficient way to address the problem. However, his GNN is trained with supervised learning, and finding an optimal dataset for this case is not a trivial task since, as it has been aforementioned, FGP-rOEE algorithm has an exponential time complexity and cannot be scaled up to circuits with a higher number of qubits. Hence, Yu Xia's thesis has demonstrated that his GNN pipeline is constrained by the limited labeled dataset, leading to scalability problems and difficulties to generalizing the problem. Hence, due to the dataset constraint, his GNN was expected to imitate FGP-rOEE's results.

On his behalf, Arnau Pastor's thesis [8] approached the same qubit-to-core mapping problem by using a feed forward feural network (FFNN) with deep reinforcement learning (DRL). The use of the latter allows to train the neural network without the need of a labeled dataset, promising then the possibility to go beyond the bounds of Yu Xia's thesis [9] and the FGP-rOEE algorithm [6]. Nevertheless, due to their nature, FFNNs only allow fixed-size inputs, which is a clear drawback because this means that the FFNN can only be used for quantum circuits with a fixed number of qubits.

Other relevant studies that use DRL for qubit routing are *Deep Reinforcement Learning Strategies for Noise-Adaptive Qubit Routing* [23] and *Practical and efficient quantum circuit synthesis and transpiling with Reinforcement Learning* [24].

3.3 Circuits of varying number of qubits

FGP-rOEE algorithm allows partitioning circuits of different sizes, but faces significant scalability problems for real-world scenarios. Yu Xia's thesis allows circuits of varying sizes but is constrained by a dataset of limited scope. Finally, Pastor's thesis can scale to larger circuits thanks to its DRL approach, but his agent can only be trained with a fixed number of qubits and cores.

Thus, this thesis leverages reinforcement learning to address dataset limitations while ensuring scalability to circuits with varying numbers of qubits thanks to the use of GNNs. It adapts both Arnau Pastor and Yu Xia's thesis to solve the graph partitioning problem applied to qubit inter-core movements. Specifically, graph neural networks are used with deep reinforcement learning. In this way, thanks to GNNs, any kind of circuit is considered (the input size is not fixed), and the model is not limited by the dataset as DRL is used.

Since every time slice of a quantum program is described as a graph (lookahead graph), where the nodes represent the qubits and the edges the interactions, it is just a graph partitioning problem, in which qubits are assigned to partitions (cores). This graph partitioning problem is then solved by our DRL environment with GNN.

Chapter 4

Methodology

This chapter details the research methodology, beginning with the fundamental configuration used in this thesis, followed by the various optimization approaches and their hypotheses, and ending with the key training approaches and their respective hypotheses. The appendix A describes the operational methodology, which focuses on the practical and logistic aspects used to conduct this work.

4.1 Fundamental Configuration

4.1.1 Dataset

The dataset to train the DRL agent consists of quantum circuits. However, at present, the existing circuits may not cover a wide enough range of applications or configurations, which presents a significant limitation for the model's ability to generalize.

On the other hand, using exclusively a dataset of random circuits prevents the model from capturing the underlying patterns of real examples. Additionally, the space of random circuits may be too large to reflect the specific structures present in real ones. Thus, in this work, a balanced dataset is used, composed of random and synthetic circuits. The latter imitate real circuits with a degree of randomness while maintaining realistic patterns. In this work, synthetic circuits are generated based on Ramos' quantum generator [25].

The scarcity of available real circuits, combined with the need for a more diverse dataset without losing real-world circuit structures, makes this balanced dataset essential for a robust and effective training.

4.1.2 Observation and Action spaces

In this section, the observation and action spaces employed in this work are described.

Observation
The environment of the deep Q-learning algorithm gives feedback continuously to the DQN agent through *observations* (states) at each time step. The observations should provide only the necessary information to the agent to select the appropriate action in that moment.

The input quantum circuits are divided into time slices, where gates are grouped and parallelized. The DRL algorithm iterates through all the time slices of a circuit. Hence, at a given time slice t, the observation should include the following relevant information:

- Current assignment of qubits to cores
- Immediate interactions between qubits in time slice t
- Future interactions between qubits in the subsequent time slices

As it has been previously approached in Sections 2.2 and 3, all this required information can be merged into a lookahead graph. Thus, in this thesis, the same idea of lookahead graph is used to describe the observations. One lookahead graph is generated per each time slice of the circuit.

The node features (core assignment) are represented as one-hot vectors. Additionally, the weight of an edge between qubits q_i and q_j at time slice t is formally represented as follows:

$$w(q_i, q_j) = \begin{cases} \infty & \text{if } q_i \text{ and } q_j \text{ interact in time slice } t \\ \sum_{t<m\leq T} I(m, q_i, q_j) D(m-t) & \text{otherwise} \end{cases} \tag{4.1}$$

where:

- $I(m, q_i, q_j)$ indicates whether q_i and q_j interact in time slice m.
- D is a monotically non-negative decreasing function.

Action
At a given observation, the agent must select an action to change the state. The elected actions are swaps between pairs of qubits, which means swapping the core assignments of qubits. This action has been considered appropriate due to its simplicity and for ensuring the fixed capacity constraint of each core.

4.1.3 Deep Q-learning algorithm

The overall algorithm iterates through every time slice t of a circuit, and at each time slice it swaps pairs of qubits until the current slice becomes valid. A time slice is considered valid when all the quantum gates in t can be executed (since each pair of interacting qubits is in the same core). Therefore, based on the current observation (state), the agent decides the next swap to do.

One of the objectives of this work (Section 1.4) is developing an agent that allows circuits of different sizes. The general idea behind deep reinforcement learning consists of making the agent select an action by means of a neural network, specifically a GNN in this study. Typically, the input to the neural network is the state, and its output dimension corresponds to the total number of actions, representing either the probabilistic distribution (in policy-based methods) or the value (in value-based methods) of each action. In this case, since the number of actions, i.e., qubit swaps, depends directly on the number of qubits, which is expected to be variable, then inevitably the number of actions become variable too. However, normally, the output dimension of a neural network (i.e., the output of the Readout function) is fixed, limiting both the number of actions and qubits. Thus, this represents a huge limitation to allowing circuits of different number of qubits.

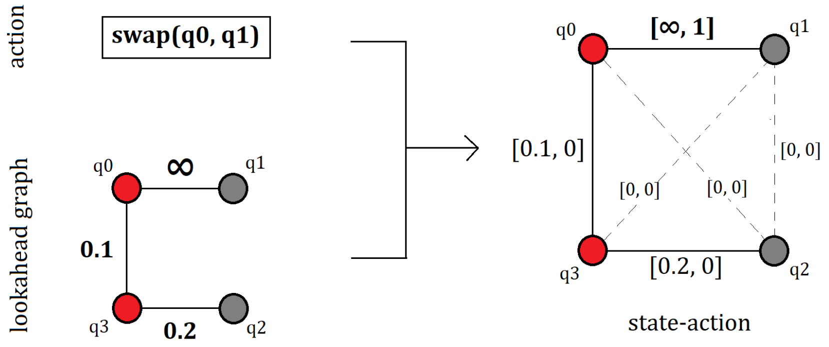


Figure 4.1: Example of *state-action* graph, at a given time slice t, with arbitrary values. The lookahead graph (*Bottom-Left*) has as node features the core each node belongs to (i.e., red or dark, which is in fact a one-hot vector), and the weights of the edges represent the interactions between qubits. The value of infinite indicates that the two involved qubits need to interact in the current time slice t. Then the values of 0.2 and 0.1 indicate that they need to interact in future slices (the higher the value, the closer is the next interaction). (*Right*) When merging *state* and *action*, a new graph is created with the same node features, and two-dimensional edge features: the first value is the weight of the lookahead graph, and the second value indicates whether the action *swap* involves the two adjacent qubits of the edge (1 if applicable, 0 otherwise).

Alternatively, if the input to the neural network becomes the combination *state-action* and the output is a single Q-value, the fixed output dimension constraint is met, allowing for forward passes on as many *state-action* tuples as actions. In this way, at a given state, a forward pass is executed per each action, providing a Q-value for each tuple *state-action*. Thus, at this point, the agent can be trained with circuits of different number of qubits.

Since the neural network in this thesis is a GNN, the input should be a graph. Therefore, the input *state-action* must be merged into a graph. Figure 4.1 depicts this fusion, which is the input to the GNN. This *state-action* graph is the observation lookahead graph with a slight modification. The edge features are no longer a weight of one dimension, a new dimension is added to represent the swap action. For each edge, the second dimension is the value 1 if the concerned qubits are the selected to be swapped or 0 otherwise.

With this approach, the number of qubits is not limited to a fixed number. In each iteration, at a given state, all the possible k actions are evaluated and, hence, all the k *state-action* graphs are generated and passed to the GNN. Then, the *state-action* with the highest predicted Q-value is chosen during the training.

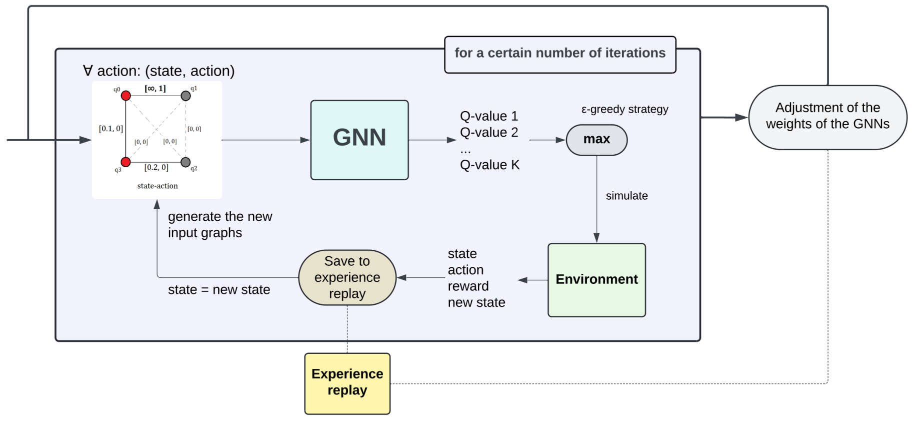


Figure 4.2: Architecture of the DQN agent

Figure 4.2 illustrates the pipeline of the DRL system.

4.1.4 Reward

In this thesis, multiple rewards have been evaluated, but the one that has proven better results is described in this section.

The rewards of the actions are calculated with the graph theory term of *cut*. The cut of a partitioned graph (in this thesis, the lookahead graph that represents the state) is the sum of the weights of the edges that have their nodes in different partitions. In the case of the state of the DQN agent, the cut is the sum of the weights of the edges across qubits of different cores.

Since the weights of the edges indicate the necessity of two qubits to interact, it is optimal that the largest weights involve qubits that are in the same core. Therefore, the agent's objective is to find a partition that minimizes the cut.

Hence, after a swap, a reward that stimulates minimizing the cut is the following:

$$\text{reward} = \text{cut}_{\text{prev}} - \text{cut}_{\text{after}} \tag{4.2}$$

where:

- cut_{prev} is the cut before doing the swap
- $\text{cut}_{\text{after}}$ is the cut after doing the swap

While this reward has been the baseline, several extensions have been analyzed and slight variations have been used, for example to stronger penalize useless actions. However, all the proven rewards have in common their aim to reduce the cut.

4.1.5 Finishing mechanisms

The agent iterates through the whole sequence of time slices of a quantum circuit. In each time slice, consecutive swaps are performed until a valid qubit-to-core assignment is reached, after which the time slice advances. When all the time slices have been iterated, the algorithm stops.

However, in a time slice, the agent might find it impossible to reach a valid assignment of qubits, causing an infinite loop of actions. In order to avoid this behaviour, a maximum number of actions is set in each time slice. If the agent reaches the maximum number of actions, the time slice is marked as invalid, and it advances to the following one.

4.1.6 Evaluation Metrics

In order to evaluate the performance of the model, key metrics are used and described in this section.

Reward

The evolution of the reward during the training is the principal indicator to analyze if the agent is learning a strategy. The objective of the DRL agent is to maximize the reward. Therefore, an upward trend in the reward means that the agent is learning, while a flat trend indicates the incapability of the agent to learn.

Invalid Slices

The percentage of invalid time slices reflects the capability of the agent to assign qubits correctly in the totality of the time slices. Thus, it is interesting that it has a downward trend during training.

Useless Swaps

It is important to keep track of useless swaps, which are actions that do not help in reaching the agent's optimal behaviour, which consists of effectively mapping qubits to cores. The desired trend is observing a decreasing trend in the number of useless swaps during the training process. Examples of useless swaps are swaps between qubits of the same core, or swaps between qubits that need to interact in the current time slice.

The reward metric indicates the learning progress of the agent, while the invalid slices and useless swaps metrics are indicators referring to the objective of this thesis, which is mapping effectively qubits to cores.

4.1.7 Evaluation methods

During the training, the agent is continuously evaluated after a certain number of iterations, where the evaluation metrics are measured. During the evaluation, the ε randomness is not considered. In this thesis, two evaluation methods have been used.

- *Highest Q-value.* The agent selects the action with the highest Q-value. It is deterministic.
- *Sampling.* The agent selects from a probabilistic distributions of actions based on their Q-values

4.2 Computational Optimization

This section first justifies why efficiency is a challenge in this work and then describes various approaches to improve the algorithm's efficiency and reduce training time.

4.2.1 Efficiency Challenge

In Arnau Pastor's work [8], for a given state, only one forward pass through a feed forward neural network is needed to predict the action to take.

However, in this project, given a state, a forward pass to the GNN is needed for each possible action. Additionally, a forward pass to the GNN is indeed multiple forward passes to the diverse neural networks that form the GNN.

The GNN, which is a MPNN, is composed of the message function (a feed forward neural network), an update function (a gated recurrent unit cell) and the readout function (a feed forward neural network).

Considering N as the number of nodes (qubits) of the input graph :

- The message neural network is used for every edge of the input graph, in both directions, which is $N(N-1)$ times.
- The update cell is used for every node (N)
- The readout neural network is used once

The messages and updates are repeated T times (hyperparameter).

Therefore, to predict an action, in [8], only one forward pass to an FFNN is needed, whereas the DQN agent needs $K(T[N(N-1)+N]+1)$ internal operations (forward passes to internal neural networks), being K the number of possible actions at the given state. Since K is the number of all possible swaps, $K = N(N-1) \div 2$. Thus, the DQN agent needs $O(TN^4)$ forward passes to select one action.

For this reason, it is crucial to create a very efficient model to try to mitigate this issue.

4.2.2 Baseline: Initial Algorithm Overview

Algorithm 1 is the pseudocode of the training DQN algorithm used as first approach.

The GNN has been implemented using the Python library TensorFlow [26]. To represent all data, this library uses a tensor data structure, which is an multi-dimensional array with a uniform type. TensorFlow uses tensors as the

Algorithm 1 Train DQN agent

```
1: env = generate_environment()
2: agent = create_DQN_agent()
3: for each episode_it from 1 to 10000 do
4:     state = env.reset()
5:     while not the end of the episode do
6:         actions = possible_actions(state)
7:         input_tensors = create_tensors(state, actions)
8:         action = agent.act(input_tensors)
9:         new_state, reward, truncated, done = env.make_step(action)
10:        save_to_experience_replay(state, action, reward, new_state)
11:        if truncated is True then
12:            mark current time slice as invalid
13:            change of time slice in env
14:        end if
15:    end while
16:    if episode_it is multiple of 20 then
17:        update_GNN_weights()
18:    end if
19: end for
```

fundamental data structure to represent and manipulate information during the training and execution of neural networks, since they are optimized to provide a high computational performance.

The different functions used in Algorithm 1 are the following:

- ***env.reset()*** resets the environment to start a new episode with a new circuit.
- ***create_tensors(state)*** creates the necessary input tensors to the GNN that represent all the tuples *state-action* for each possible action at the given state.
- ***agent.act(input_tensors)***. The input tensors are passed to the GNN so that the agent decides which action to take based on an ε-greedy strategy. In the Python code, *possible_actions(state)* and *create_tensors(state, actions)* are, in fact, inside the *agent.act()* function, but for readability reasons, they are shown outside of this function in Algorithm 1.
- ***env.make_step(action)*** simulates in the environment the choosen action by the agent. It returns the new state after having applied the action, the

immediate reward, and two Booleans indicating whether the episode has finished (done[1]) or truncated[2]. Both Booleans are false otherwise.

- ***save_to_experience_replay(state, action, reward, new_state)*** saves the state, the action, the reward of applying the action to the state, and the new state to the experience replay. Later, the agent will take samples from the experience replay to train the GNN.
- ***update_GNN_weights()***. Every 20 episodes (this number can be changed), the agent trains the GNN using samples from the experience replay. This process is analogous to determining batch size, as the agent accumulates multiple episodes to ensure a consistent amount of data for training.

Considering this algorithm is essential for further efficiency enhancements that will be addressed subsequently.

4.2.3 Identification of Bottleneck

In order to improve the efficiency, it is crucial to find the potential bottleneck.

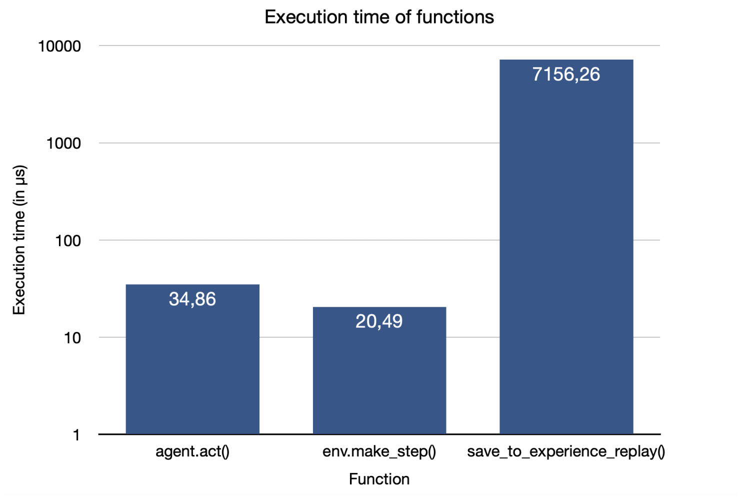


Figure 4.3: Execution time of functions (in μs)

The most repeated part of Algorithm 1, and thus the one that may have the highest computational weight, is the *while* loop, which executes the process

[1]An episode is considered *done* if all qubits have been correctly assigned to cores in all the time slices of the circuit. The episode is, hence, considered as finished.

[2]In a time slice, when the number of actions (swaps) has reached a specific maximum number without being capable of making the slice valid, then the time slice is considered as *truncated* and no more actions are taken in that time slice. This is done to avoid infinite loops of actions.

of choosing the most appropriate action, simulates it in the environment, and saves the *state, action, reward and new state* into the experience replay. Figure 4.3 depicts the execution times of the three main functions (in logarithmic scale) used in the *while* loop. It is evident that the function *save_to_experience_replay()* is a clear bottleneck, as it is around 129 times slower than the other two functions together. It represents a 99% of the total execution time. Therefore, the focus must be put now in this function.

Internally, *save_to_experience_replay()* calls to *create_tensors()* twice to create the input tensors of two forward passes: $Q(s, a; \theta)$ and $\max_{a'} Q(s', a'; \theta^{-})$ (in order to compute the loss function, as aforementioned in Section 2.3.5). Thus, optimizing *create_tensors()* may improve significantly the total execution time.

4.2.4 Optimization 1: Duplicated Messages removed

In this thesis, in the MPNN, the Message function, i.e., a feedforward neural network, is formally defined as $M(v, w, e_{vw})$, being v, w two nodes and e_{vw} the edge feature between those two nodes.

Since the used graphs are undirected, for each pair of nodes (v, w), two messages are computed: from v to w, and from w to v, which are the same in both cases. Therefore, for each pair of nodes, it is only necessary to calculate one message instead of two, and store it in order to be reused. In this optimization, the tensors created by the function *create_tensors()* and the message calculations within the the MPNN are cut in half.

Hypothesis. *save_to_experience_replay()* represents a 99% of the total execution time, and it is primarily composed of two calls to *create_tensors()*. Hence, reducing the number of messages by half (removing half of the input edges) will halve the input tensors generated by *create_tensors()*. Therefore, by removing duplicated messages of the edges, it is expected to reduce the execution time by half.

4.2.5 Optimization 2: Useless Tensors removed

create_tensors() has potential for further optimization. This function generates many auxiliary tensors that are used to create the resulting ones. Tensors are useful and optimal when used in a @tf.function[3]. However, during the process of creating tensors for the GNN, generating unnecessary intermediate tensors can overload computations, since it is costly to generate tensors that are not used efficiently at a later stage.

[3]A @tf.function is a function compiled into a callable TensorFlow graph that can compute with tensors in an optimized way, using specific parallelizable functions. It is commonly used when operating with neural networks.

During the process of generating the input tensors, an intermediate operations involves sequentially iterating through a list of actions, which is previously transformed into tensor. In this case, no parallelism is used. Thus, there is absolutely no need for transforming the auxiliary list of actions (*swapList*) into a tensor.

```
actions = tf.constant(swapList)
...
embedded_indices_k =
    [[offset*num_all_edges + f(i,j,len(node_states))]
    for offset, (i, j) in enumerate(actions)
    if (i,j) != (-1,-1)]
```

According to the code above, `tf.constant(swapList)` should be changed to `swapList`. This data structure is no longer interpreted as a tensor and simplifies the tasks.

Although this may seem a little change, *swapList* is affecting a process that is being executed an very large number of times and its data is repeatedly used throughout the code.

Hypothesis. By removing unnecessary tensor creations from the function *create_tensors()*, the total execution time will improve.

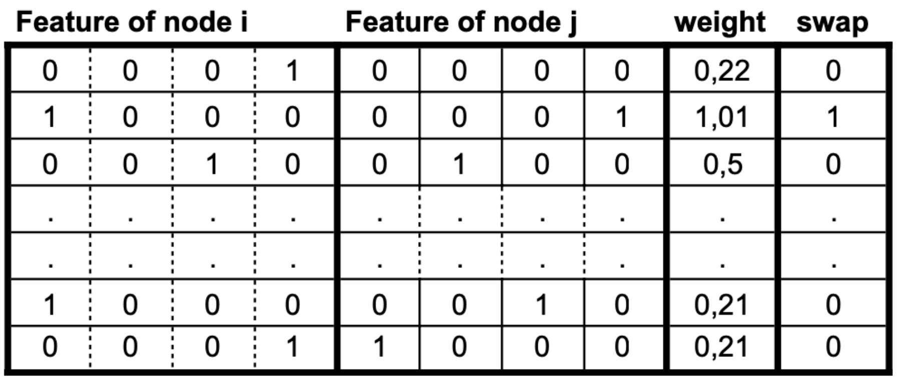
4.2.6 Optimization 3: In-place Tensor Modification for State Transitions

Selecting an action is the operation that is in the innermost loop of the overall algorithm, in which the input tensors for the GNN are created based on the current state and all the possible actions. Creating these tensors is a computationally expensive task. Thus, reducing their creation to the minimum number of times could potentially increase the efficiency of the algorithm.

The input tensor to the GNN is a 2-dimensional array, where each line represents an edge of the input graph (composed by the node features of the two adjacent nodes, and the edge features), and all graphs (one graph per action) are represented.

The columns define the features of the two adjacent nodes and the ones of the edge, as mentioned above. The features of a node, which represents a qubit, indicates the core to which it is assigned (one-hot vector), as aformentioned in Section 4.1.2. The features of the edges (seen in Section 4.1.3) have 2 dimensions: the weight of the edge, and the swap value (0 or 1).

Figure 4.4 depicts the structure of the input tensor to the GNN.



Feature of node i				Feature of node j				weight	swap
0	0	0	1	0	0	0	0	0,22	0
1	0	0	0	0	0	0	1	1,01	1
0	0	1	0	0	1	0	0	0,5	0
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.
1	0	0	0	0	0	1	0	0,21	0
0	0	0	1	1	0	0	0	0,21	0

Figure 4.4: Input tensor structure

Instead of creating this tensor every time, it is more efficient to create a unique tensor at the beginning of the episode, then every time the state changes modify directly this tensor.

In this case, when a swap is made between qubits q_i and q_j, the state changes, since the core assignment is swapped between those two qubits. Thus, in the tensor, the lines (edges) that involve one or both of these qubits, need to be changed by updating the node feature.

Furthermore, during the execution of the episode, if the time slice changes, the weight of the edges too, so the column of weight in the input tensor needs to be changed.

Thanks to this practice, instead of creating a whole new tensor every time the agent has to decide an action, only the necessary lines are modified in the existing tensor.

Hypothesis. Every time the observation state changes, modifying the existing input tensor to the GNN instead of creating a new one will significantly improve the execution time.

4.2.7 Optimization 4: Parallelization of Independent Episodes

Each episode, which is the sequence of actions (swaps) taken through all the time slices of a circuit to assign qubits to cores, is completely independent. Overall, the algorithm runs various episodes to store in the experience replay tuples of *state-action-reward-new_state* in order to use them in the future to update the weights of the GNN. Since the episodes are independent, they can be executed in parallel.

In order to take the maximum profit from the parallelization, it is optimal to run one episode per classical core of the computer. Therefore, every core will run one independent process.

Nevertheless, there is only one common operation done by all the episodes that is not independent: storing to the experience replay. The latter is a data structure shared by all the episodes, to which every process needs to access. This could lead to a race condition.[4]

A first solution could be controlling the access to the experience replay by blocking any process to write while another is accessing the shared data structure. However, this solution will create a queue of processes waiting to write into the experience replay sequentially, which would lead to an important bottleneck.

Therefore, a better approach would be creating a private memory for each process. In this way, all the experiences will no longer be stored in a shared memory, but in a local memory of each core instead. Then, at a later stage, when the algorithm will need to access the experience replay to update the GNN, it will randomly pick samples from the different local memories of the cores. In other words, the experience replay will no longer be a unique shared memory, but rather a distributed memory among cores. Figure 4.5 illustrates this idea.

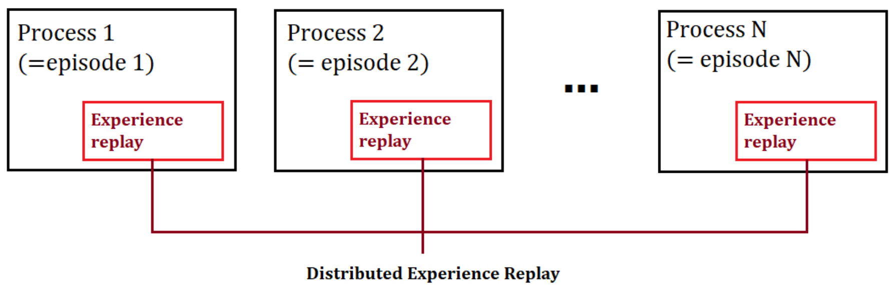


Figure 4.5: Distributed experience replay among N cores

Hypothesis. Parallelizing the algorithm across multiple cores is expected to reduce execution time proportionally to the number of cores.

[4]A race condition occurs when the behaviour of a program depends on the sequence or timing of uncontrollable events, such as concurrent access to a shared resource, which may lead to data corruption or unpredictable results.

4.3 Training Experiments

In this section, the training approaches that have performed the most significant results are described. The methodology employed involves incrementally increasing the task difficulty, beginning with simple problems (i.e., small architectures) and progressively advancing to more complex challenges.

While during the same training diverse circuits of different number of qubits can be trained, the approaches are based on a fixed number for a clearer analysis of the approach. The values of the rewards are not compared in detail, as the reward function differ slightly across approaches, as outlined in Section 4.1.4. Analyzing their trend is what describes the agent's learning ability.

4.3.1 First Configuration: 4 Qubits and 2 Cores

First, the simplest problem is analysed, which consists of allocating 4 qubits in 2 cores. In this case, the model is evaluated throughout the training process by selecting the action with the highest Q-value.

Hypothesis. With this simple setup, it is expected that the agent will learn and converge rapidly due to the simplicity of the problem. Furthermore, the number of invalid slices should converge to 0.

4.3.2 Second Configuration: 6 Qubits and 3 Cores

The problem size is increased to analyze how the agent performs with more difficult challenges. At this stage, the agent works with circuits of 6 qubits and 3 cores. In this configuration, the two evaluation methods are compared: *Highest Q-value* and *Sampling*.

Highest Q-value Evaluation Hypothesis
Similarly to the first configuration, the number of invalid slices will converge to a slightly greater number of invalid slices compared to the first approach, since the complexity of the problem is higher. The agent is expected to learn; thus, the accumulated reward should increase while the number of invalid slices decrease.

Sampling Evaluation Hypothesis
When evaluating through *sampling*, it is highly possible that the selected action is not the most optimal (the one with the highest Q-value), but it may allow the model to escape from suboptimal or repetitive behaviours. A wider range of actions might be selected, causing the model to avoid inconsistent peak rewards or patterns. Thus, due to its lack of determinism, it might be slightly more difficult to observe a significant upward trend in the learning process, but a smoother learning (reward) curve instead. The *sampling* approach should

help overcome stagnation behaviours, if present, leading to better results in the invalid slices metric.

4.3.3 Third Configuration: 8 Qubits and 4 Cores

At this point, circuits of 8 qubits and 4 cores are trained. In this case, masks are applied to the model and further analyzed. The model is evaluated with the method *Highest Q-value*.

No Mask Approach
First, the model is trained with no masks, which means that any action can be selected. The reward used in this case is the difference of cuts ($\text{cut}_{\text{prev}} - \text{cut}_{\text{after}}$), and explicitly it strongly penalizes useless swaps (e.g., swaps within the same core). On the other hand, it gives a positive reward to swaps that help to reduce the cut.

Hypothesis. Penalizing useless actions and rewarding useful ones is expected to make the agent learn to select the beneficial actions without need of masking.

Mask of Useless Actions Approach
In this approach, a mask is used to avoid selecting actions that are useless.

- *Last Action.* Repeating the last action makes the model go back to the previous observation state. By doing the same action twice, the definitive state does not change. Therefore, it is useless.
- *Swaps within the same core.* Swapping qubits within the same core does not change the state, since qubit allocations remain unchanged.
- *Swaps between interacting qubits.* Swapping qubits of different cores that are involved in the same qubit gate in the current slice does not allocate them in the same core.

With this mask, the agent does not explore through useless actions. Since the evaluation is deterministic (*highest Q-value*), applying an action that does not affect the state would make the agent select the same action in the subsequent step, since the input to the GNN would not change; hence, neither the selected action.

Hypothesis. By adding a restrictive mask to avoid useless actions, the agent will be guided in its learning process. This change may result in a worse learning performance, since part of the learning has been accomplished explicitly with the mask. However, the results of invalid slices should be better as the agent avoids exploring unproductive paths.

Chapter 5

Results

In this section, the results of the different approaches described in Section 4 are analysed. First, the efficiency approaches are analysed and compared. Subsequently, the training results are discussed.

5.1 Computational Optimization

In order to improve and reach feasible training times, the five computational approaches described in section 4.2 are analyzed.

- **Baseline:** Initial algorithm
- **Optimization 1:** Duplicated messages removed
- **Optimization 2:** Useless Tensors removed
- **Optimization 3:** In-place Tensor Modification for State Transitions
- **Optimization 4:** Parallelization of independent episodes

For all these optimizations, the training time is measured. In cases where direct measurement is unfeasible, the total training time is predicted proportionally based on the initial performance. The individual absolute training times are essential for achieving a feasible training duration with the hardware used in this thesis. However, training times are generally dependent on the hardware employed. Therefore, the analysis of this metric is focused on the evolution of the training times across the different approaches, which is generally independent of the specific hardware, as improvements are expected to be proportional across different systems.

Figure 5.1 depicts the evolution of the training times after applying the optimizations. It is clear to see the significant improvement in efficiency. The results are now explained in detail.

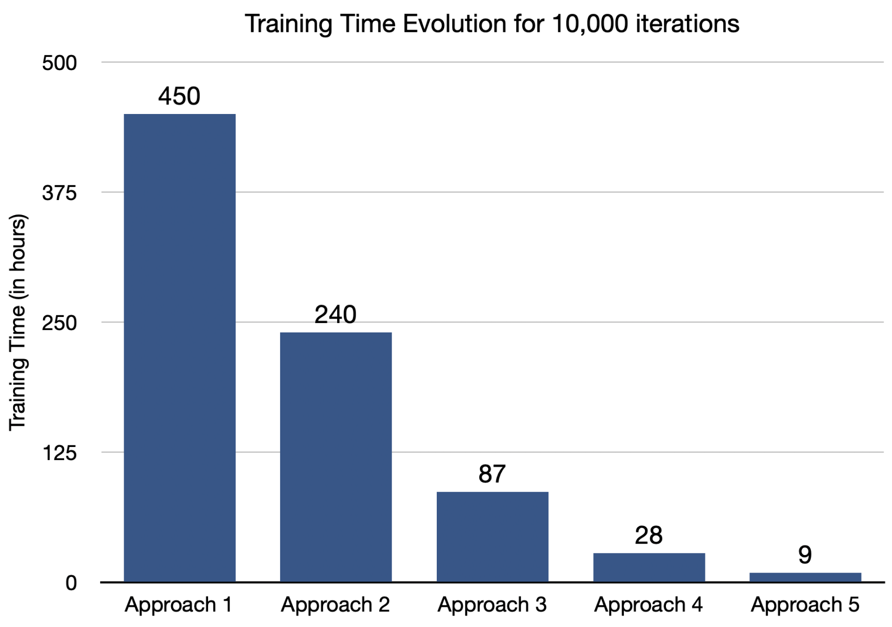


Figure 5.1: Evolution of the training times across the approaches for 10000 episodes

Baseline: Initial algorithm
In the baseline, the measured performance is 50 episodes run in 2 hours and 15 minutes. As shown in Algorithm 1, the training is considered as finished after completing 10000 episodes. Therefore, the predicted training time is (10000 ep $\times$ 135 min) $\div$ 50 ep $\approx$ 27000 min $\approx$ 450 hours.

The following optimizations try to minimize this initial duration in order to make it more viable.

Optimization 1: Duplicated messages removed
Removing the duplicated messages from the GNN is expected to halve the training time.
After this modification, the agent performs 80 episodes in 115 minutes. Which means that the predicted training time is approximately (10000 ep $\times$ 115 min) $\div$ 80 ep $\approx$ 14375 min $\approx$ 240 hours.

This significant reduction in training time, which is approximately half of the previous one, confirms the hypothesis made in Section 4.2.4. The halving of training time is a crucial advancement, as it underscores the potential of this optimization to drastically improve the scalability and efficiency of the agent's learning process. These results represent a major step forward in enhancing the overall performance of the model.

Optimization 2: Useless tensors removed
As stated in the hypothesis of section 4.2.5, removing the creation of useless tensors that are generated very frequently in the algorithm is expected to have a high impact in reducing the training time.
The impact of this change has been a reduction to 50 episodes run in 26 minutes, which is (10000 ep $\times$ 26 min) $\div$ 50 ep $\approx$ 5200 min $\approx$ 87 hours.

This modification has led to a remarkable reduction in training time, with an improvement factor of 2.75, which provides a clear evidence that the hypothesis in Section 4.2.5 has been strongly validated. The substantial decrease in execution time highlights the critical role of removing useless tensors, not only confirming the expected impact but also demonstrating a key optimization that significantly accelerates the agent's performance. These results mark a pivotal achievement in improving the overall efficiency of the algorithm.

Optimization 3: In-place tensor modification for state transitions
For each state transition of the DRL environment, modifying the existing input tensor to the GNN instead of creating a new one has a high potential to hugely improve the efficiency of the algorithm (Section 4.2.6).

The in-place modification of tensors during state transitions has resulted in a substantial improvement, reducing the training time to just 28 hours for 10000 iterations. This corresponds to a notable speedup of 3.10, confirming the high potential of this optimization as hypothesized in Section 4.2.6. The results demonstrate a crucial enhancement in the algorithm's efficiency, making training times far more feasible for large-scale applications. While 28 hours of training is viable, it can be further optimized with the fourth optimization.

Optimization 4: Parallelization of independent episodes
As aforementioned in the hypothesis of Section 4.2.7, it is expected that by parallelizing the episodes, the speedup will be proportional to the number of cores. In this case, the model has been trained in a shared Ubuntu server at the University (UPC) using 8 physical cores. The training time has been 9 hours, which is 28 hours $\div$ 9 hours $\approx$ 3.11 times faster.

While the efficiency has improved, this enhancement is not aligned with the initial hypothesis, since it was expected to be approximately 8 times faster (the number of cores). Several factors contribute to this result. The server is shared with other processes; thus, different processes are running concurrently, which inevitably impacts the performance. Moreover, not all the program is parallelized. When the episodes end, all the processes are closed and the main thread of the program continues by updating the weights of the GNN. Therefore, parallelizing does not affect the entire program.

Nonetheless, with this fourth and last optimization, the agent has finally reached a level of efficiency that is feasible to train.

5.2 Training Experiments

In this section, the training experiments presented in Section 4.3 are executed, and the useful resulting metrics are illustrated and analyzed.

5.2.1 First Configuration: 4 Qubits and 2 Cores

The first configuration is based on the simplest problem form to ensure that the DQN agent converges well to the expected behaviour.

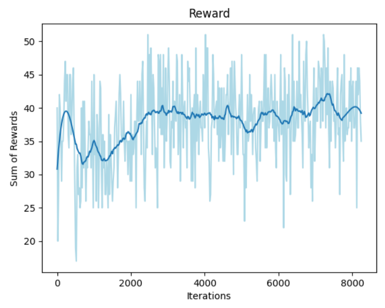


Figure 5.2: Episode cumulative reward metric for 4 qubits and 2 cores

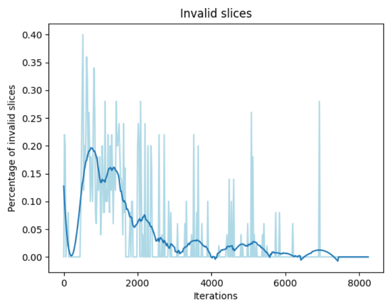


Figure 5.3: Invalid slices metric for 4 qubits and 2 cores

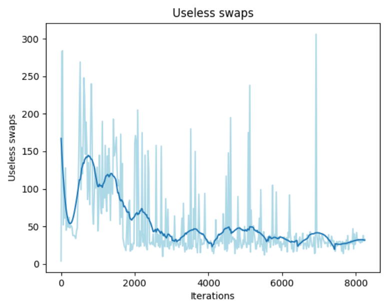


Figure 5.4: Useless swaps metric for 4 qubits and 2 cores

Figure 5.2 illustrates the cumulative reward per each episode during the training. Its observable increasing trend confirms that the agent effectively learns throughout the training iterations, until reaching a point where it converges. Concurrently, the number of invalid slices (Figure 5.3) decreases until reaching the ideal number of 0 invalid slices. Finally, Figure 5.4 shows that the number of useless swaps progressively decreases too. All these metrics indicate that the agent learns while accomplishing at the same time the thesis' goals (Section 1.4).

5.2.2 Second Configuration: 6 Qubits and 3 Cores

In this configuration, the problem size has been increased to analyze how the DQN algorithm scales with larger circuits.

Highest Q-value Evaluation Approach

First, the results with the *highest Q-value* evaluation approach are discussed.

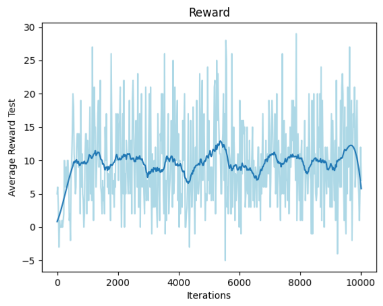


Figure 5.5: Episode cumulative reward metric for 6 qubits and 3 cores, with *highest Q-value* evaluation approach

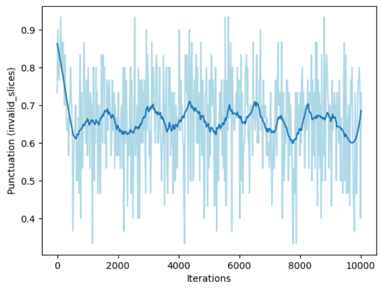


Figure 5.6: Invalid slices metric for 6 qubits and 3 cores, with *highest Q-value* evaluation approach

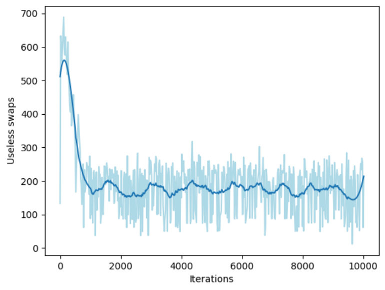


Figure 5.7: Useless swaps metric for 6 qubits and 3 cores, with *highest Q-value* evaluation approach

Figure 5.5 depicts an increasing trend in the reward function during the initial episodes of the training. However, this rise converges quickly. At the same time, based on Figure 5.6, the percentage of invalid slices decreases from approximately 85% until reaching the stagnant range of 60-70%. While it is a positive signal that the number of invalid slices decrease, a 60-70% of invalid slices is a high number. It was expected that this number was slightly greater than 0, but it is much larger than initially expected.

At the same time, there is a logical correlation between a reduction of the number of useless swaps (Figure 5.7), a decreasing trend in the percentage of invalid slices, and the increase of the reward (the agent learns). While the trends of the metrics are promising, the final values get stuck earlier than expected, which may be a symptom of the model becoming trapped in a local optimum.

Sampling Evaluation Approach

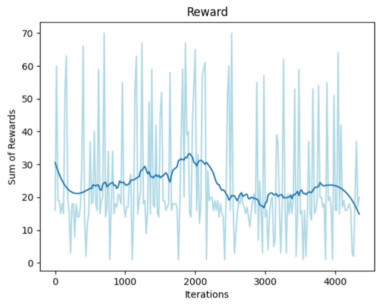


Figure 5.8: Episode cumulative reward metric for 6 qubits and 3 cores, with *sampling* evaluation approach

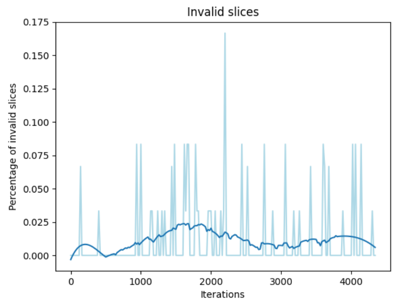


Figure 5.9: Invalid slices metric for 6 qubits and 3 cores, with *sampling* evaluation approach

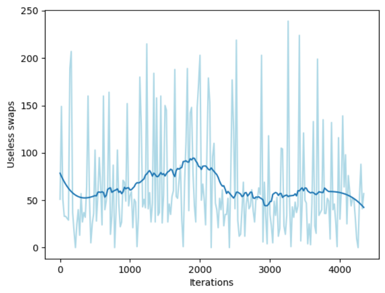


Figure 5.10: Useless swaps metric for 6 qubits and 3 cores, with *sampling* evaluation approach

By changing to the *sampling* evaluation approach, it is expected a smoother learning progress, while better results in the percentage of invalid slices (hypothesis in Section 4.3.2.

In the previous approach, the percentage of invalid slices get stuck at 60-70%, which is an elevated number. However, with the *sampling* evaluation approach, Figure 5.9 illustrates that this percentage has now decreased to around 0-2.5%, representing a significant improvement, as it was expected. Similarly, the number of useless swaps (Figure 5.10) is also significantly lower than the previous approach.

However, none of the three Figures 5.8, 5.9 and 5.10 exhibit a consistent trend in either direction, upward or downward. This means that, although the results are better, in this case the agent fails to sufficiently differentiate the Q-values, resulting in values that are too close to one another, rather than a clear separation that would indicate the agent's intended actions. This leads to a relatively uniform distribution of Q-values, obscuring the agent's decision-making process if a *sampling* evaluation method is used. Nevertheless, as aforementioned, the performance of the agent in terms of number of invalid slices and useless swaps is significantly better compared to the previous approach.

5.2.3 Third Configuration: 8 Qubits and 4 Cores

In this section, the results of circuits with 8 qubits and 4 cores are analyzed, with the *highest Q-value* evaluation method. As described in Section 4.3.3, in this more complex challenge, the impact of masks on actions is analyzed and compared.

No Mask Approach

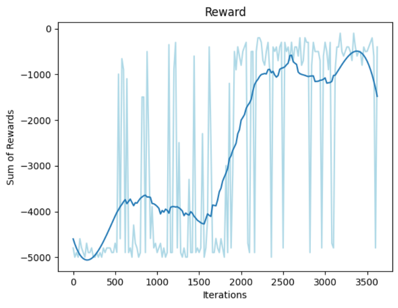


Figure 5.11: Episode cumulative reward metric for 8 qubits and 4 cores, with *no mask* approach

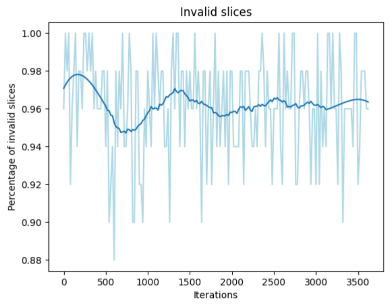


Figure 5.12: Invalid slices metric for 8 qubits and 4 cores, with *no mask* approach

As a first approach with this configuration, by penalizing useless actions and rewarding the positive ones through the reward function, without masking them, it is expected that, during the training, the model will learn and decrease the number of invalid slices, which is the hypothesis presented in Section 4.3.3.

Figure 5.11 presents a clear upward trend in the reward throughout the training phase. However, Figure 5.12 depicts a very high percentage of invalid slices (approximately 96%), which is a significantly unfavorable result.

At the beginning of the training, it has been detected that a very large number of swaps within the same core (which are explicitly penalized by the reward) are done, leading to the the accumulation of large negative reward values. As described in Section 4.3.3, the reward for non-useless swaps is the difference of cuts ($\text{cut}_{\text{prev}} - \text{cut}_{\text{after}}$). Thus, performing the same swap repeatedly leads to an accumulated reward of 0. This happens because repeating the same action involves adding and removing continuously the same absolute value to the cut. Hence, the agent learns to avoid swaps within the same core, but instead repeating the same action yields better cumulative rewards (closer to 0), which is more optimal than executing the initial unnecessary swaps. The agent adopts a strategy of repeating actions rather than prioritizing useful ones. While this behaviour makes sense, it is not the expected one, since the agent is not selecting the beneficial actions.

This repeating behaviour can be generalized to a global problem in RL, which is the *effective horizon* [27, 28]. Simplified to this case, the agent prioritizes accumulating rewards that sum to zero over risking longer sequences of swaps that may yield lower rewards in some instances but ultimately lead to a higher cumulative reward. This behavior is attributed to the agent's short effective horizon (number of future steps the agent can critically assess to achieve meaningful rewards).

The conclusion made by this approach is that, although the agent learns a strategy, the incentives of the agent are not aligned with this thesis' goals.

Mask of Useless Actions Approach

As a result of the previous incentives of the agent, a mask of useless actions is applied to the agent. It is expected that this guidance through its training will make its learning converge quicker and easier, and provide better metric results.

The observed reward in Figure 5.13 does not increase, which means that the agent is not capable of learning. By removing explicitly from its action space actions that are useless, the agent is not able to find deeper strategies to solve the problem. If the qubit mapping was effectively done, the inability of the agent to learn would be justified as there would be no more room for improvement. However, this is not the case since Figure 5.14 depicts a high percentage of invalid slices (75-80%).

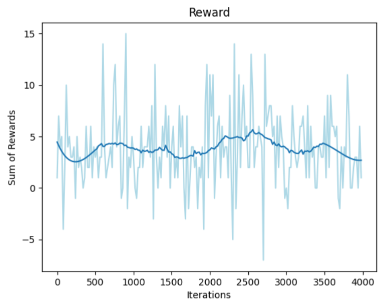


Figure 5.13: Episode cumulative reward metric for 8 qubits and 4 cores, with *mask of useless actions* approach

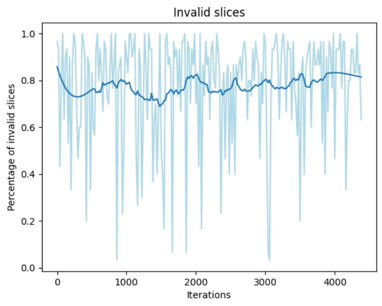


Figure 5.14: Invalid slices metric for 8 qubits and 4 cores, with *mask of useless actions* approach

Overall, while a decline in learning performance compared to the *no mask* approach was expected, the agent finally failed to learn altogether. Additionally, as anticipated, the number of invalid slices has decreased after implementing the mask. Nevertheless, this number has not decreased sufficiently to reach the ideal of 0.

Chapter 6

Conclusions

In this work, a graph reinforcement learning approach has been implemented to address the qubit mapping problem in multi-core quantum architectures, with the added value of allowing circuits of different number of qubits.

As initially expected, severe efficiency issues have been faced. However, they have been more critical than expected, which has inevitably led to an exceedingly higher number of devoted hours to this matter, at the expense of fewer available time to conduct training experiments. Regarding the efficiency approaches, they have substantially decreased the training times to reasonable levels: from 450 hours to 9 hours for 10000 episodes.

Despite these limitations, the preliminary results indicate further potential, though the measured metrics remain below expectations. The observed trends suggest that the agent is able to learn useful patterns, which means that, with a deeper dedication, the performance could significantly improve. Moreover, in line with the stated objectives in Section 1.4, compared to previous studies, the agent can be trained with circuits of a varying number of qubits.

Deeper investigation to reducing the number of invalid slices to zero opens the door to further improvements, where, once all time slices are valid, the principal aim is reducing the number of inter-core communications.

A conclusion that can be drawn is that allowing the agent to explore the full action space enhances its learning capacity, though at the cost of reduced performance. Conversely, masking actions improves performance but limits the agent's learning ability.

In summary, while this work presents modest initial results, the foundations laid by this study provide a starting point for future research, where the combination of deep reinforcement learning techniques and graph neural networks can continue to be explored to solve complex problem in quantum computing.

6.1 Future work

The approach presented in this thesis can be further explored by doing a more extended training experimentation. While this approach is not the most computationally optimal, for small circuits it is feasible to have reasonable training times. Scaling up the number of qubits increases the number of actions in the order of $O(n^2)$, hence the necessary number of forward passes. However, parallelising these forward passes (one per action) would potentially provide reasonable training times.

Furthermore, a better hyperparameter tuning or redefining the architecture of the GNN can enhance results. Future studies could also consider the use of other DRL algorithms (e.g., *Proximal Policy Optimization* (PPO)) with GNNs. The flexibility of GNNs allows to reach approaches where the number of qubits and cores are not fixed. A shift in perspective can be applied by using alternative observation representations with different graph models of the problem, or by modifying actions, such as using qubit-to-core assignments.

Appendix A

Operational methodology

This work is related to the research towards an optimal DRL model capable of solving the partitioning problem applied to qubit inter-core movements. As it is not possible to know how to model the optimal architecture in advance, and it is highly probable to change the approaches as the project progresses, there is a need for a flexible methodology that allows to continuously implement and evaluate models in order to discard them, or to open new research paths.

The Agile methodology is convenient for this purpose, since it offers flexibility and well-suited practices for projects where learning and exploration are integral parts of the process.

Meetings with the tutors are done weekly to update the situation of the project, and propose new objectives or redirect tasks if necessary.

A.1 Tools

The development of the code is managed by Git, a distributed version control system, to keep every version of the code saved and rollback if it is necessary, since this research involves the use of different approaches to tackling a problem, trial and error, experimentation, hence the use of Git is suitable for this case. Particularly, the repository used is GitHub so that concerned people can also have access to it.

Appendix B

Temporal planning

In this section, different tasks are detailed, their dependencies are identified, and a temporal planning for them is developed in order to accomplish the objectives of this project, described in Section 1.4. It is crucial to have it all arranged from the beginning to finish this thesis within the established time frame.

The start date is the 12th of February, and it is expected to end maximum on 18th of June. Thus, overall, the project is expected to take approximately 128 days, which corresponds to around 18 weeks.

According to the UPC's regulations, the Bachelor's thesis should take 540 hours of work, which needs to be spent throughout the aforementioned weeks. The expected dedication of time is around 30 hours per week: on weekdays approximately 20 flexible hours are devoted to the project, then 10 more hours during the weekend to write and improve the report.

B.1 Tasks

This work has been divided into different sub-tasks grouped into 4 main groups: Previous Study (PS), Documentation (DOC), Implementation of the DRL Agent (DRL), Evaluation & Comparison (EC).

A more detailed description of them is presented in the following sections.

B.1.1 Previous Study (PS)

- ***PS1*: Quantum context.** The use of Artificial Intelligence in this project is aimed to be used in a quantum context. Therefore, in order to make this research succeed, it is necessary to previously understand this realm, i.e., dive deep into quantum multi-core architectures, quantum gates, and quantum programs.

- ***PS2*: Deep Reinforcement Learning.** Understand what reinforcement learning is, how it leads to deep reinforcement learning, the different architectures, and the influence of the hyperparameters to the model's performance. Study how previous studies have used DRL to solve similar problems.
- ***PS3*: Graph Neural Networks.** GNNs have already been used to solve similar graph partition problems, so this task consists of analysing those studies and considering how they can be adapted to the objectives of this thesis (Section 1.4). Before, it is necessary to learn about GNNs, their possible architectures, the used hyperparameters.
- ***PS4*: Merging GNN with DRL.** Once the use of GNNs and DRL have been investigated in depth, it is time to think about how to merge GNN with a DRL agent to solve the problem of this thesis (Section 1.3).

Since the topic of this research has been totally new for me, a considerable number of hours will be spent to study the quantum context, and to understand well how GNNs and DRL can be applied in this domain. Therefore, overall, two weeks (40 hours) of previous study are expected.

B.1.2 Documentation (DOC)

- ***DOC1*: Contextualization and definition of the Scope.** Write the first part of the report of the thesis, which involves writing an Introduction, the context (i.e., quantum, GNN and DRL), defining the problem and the scope, then the methodology that will be used.
- ***DOC2*: Temporal planning.** Resolve the temporal planning of the project by identifying different tasks, their dependencies and their duration. Represent the planning through a Gantt diagram and a task summary table.
- ***DOC3*: Economic Management & Sustainability.** Think about the economic dimension of the project and describe in detail the budget for it based on the tasks, estimations and management control. Furthermore, write an analysis about the sustainability matter of this project.
- ***DOC4*: Documentation of the development of the project.** As the project advances, the discoveries and results will need to be documented in the report. This task is expected to be done in parallel with the project weekly.

The report represents the heart of this project, so it is essential to devote the deserved time to it. The estimated time is spending 10 hours per week to continuously write and improve the report (170 hours in total). This will be done during the weekends in order to summarize the weekly advances in the report. However, as this project is flexible and the workload may vary, the amount of time spent for the documentation may change when necessary.

B.1.3 Development of the DRL agent (DRL)

- ***DRL1*: Design the state space.** Define the characteristics that constitute the state space of the DRL environment, considering relevant information about the quantum program represented as a graph, such as qubit interactions, qubit-to-core assignments and time slices.

  In the DRL agent, the way the states will be represented will significantly influence in the results of the model, so choosing the state space must be done with special emphasis. Hence, it is expected to take a week (20 hours) to properly design the state space, time enough to consider different types of graphs to represent the state of a quantum program. However, as this part is completely exploratory, this number of hours is susceptible to change.

- ***DRL2*: Design the action space.** The DRL agent will need a set of actions to perform and learn, hence, this task consists of designing adequate actions to solve the graph partitioning problem.

  The actions have not too much complexity compared to the state space, so the devoted time to designing and implementing the actions will be around 2-3 days, which corresponds to 8-12 hours. Nevertheless, if there is time to kill, more actions could be explored.

- ***DRL3*: Integration with GNN.** Integrate GNN within the DRL agent. This includes designing the architecture of the GNN and deciding the input graph form which best describes the quantum program state in a given time slice. Consider also different approaches to the graph representation to compare the results in the future.

  This part may lead to problems, so the time for it will be one week (20 hours) to guarantee that the fusion will be made with no further problems.

- ***DRL4*: Develop reward structure formulation for the DRL agent.** Define the reward function which gives feedback to the DRL agent. To this aim, it is vital to determine the objectives and constraints of the qubit inter-core movement partitioning problem to define a reward scheme that stimulates the desired behaviour of the model.

  The DRL agent's performance will be highly influenced by the reward function, so it is important to set an adequate reward. Therefore, a week (20 hours) will be dedicated to this matter.

- ***DRL5*: Design termination conditions for the DRL agent.** Define and implement the conditions to end an episode of interactions between the DRL agent an the environment, such as a maximum number of steps or predefined stopping conditions.

- ***DRL6*: Implement the DRL environment dynamics.** Finish to implementing all the necessary dynamics of the environment to interact correctly with the DRL agent.

  Programming the dynamics of the environment and the termination conditions (*DRL5* and *DRL6*) will take also a week (20 hours), to finish setting up the whole environment and the agent to work as a whole.

B.1.4 Evaluation & Comparison (EC)

- ***EC1*: Baseline evaluation using FGP-rOEE algorithm.** Implement the FGP-rOEE algorithm [6] that will be used in the future to compare it with the DRL agent's performance in terms of scalability, computational efficiency and results. This task is necessary to accomplish the objectives of this project (Section 1.4).

  This will take around 4-5 days, which corresponds to 16-20 hours.

- ***EC2*: Set up synthetic and real circuits.** Create the dataset to train and test the model. Create and collect both synthetic and real circuits to train and evaluate the model later on.

  It will need first a little of research to deduce how to produce the circuits, then implement a program that creates the required circuits. Moreover, real circuits will need to be chosen to test the agent. Overall, this workload is estimated to take also one week (20 hours).

- ***EC3*: DRL agent training.** For each different approach, train the DRL agent using techniques to find the best adjustments of the hyperparameters of the model. If necessary, fine tune the model.

- ***EC4*: Evaluate and Compare.** Compare the best performing approaches of the DRL agent with the FGP-rOEE algorithm [6] to evaluate the overall performance. Then, evaluate and draw conclusions of the DRL agent, checking if the objectives of the project (Section 1.4) have been accomplished.

  As tasks *EC3* and *EC4* are highly related, the estimated time can alternate between them, so they are considered as a whole. The training phase may take very long depending on the complexity of the model, and since there will be several trainings to adjust the hyperparameters and to compare approaches, this phase is expected to take too much time. That is why, the time devoted to it will be of 150 hours, however it is hard to estimate time in this case because it can vary considerably. As this represents the last run, these hours will be compressed within a shorter period of time to finish the project with some margin to have flexibility to solve any possible setback, explained in Section D.

B.1.5 Meetings (M)

In order to monitor the progress of the project, different meetings are scheduled with the supervisor and co-supervisor of this thesis. They have been divided into 4 different types: initial meeting, regular monitoring meeting, intermediate evaluation meeting, and the final meeting.

At least one face-to-face meeting will take place every week with the expected duration of one hour. Thus, a total of around 18 hours (18 weeks) will be dedicated to meetings.

- ***M1*: Initial meeting.** Initial meeting to introduce the topic of the thesis and get initial information about the project.
- ***M2*: Regular monitoring meeting.** Weekly, regular monitoring meetings take place in order to present the results of every sprint (following the Agile methodology explained in Section A) or to simply resolve doubts about the project.
- ***M3*: Intermediate evaluation meeting.** As stated in the regulations of this Bachelor's thesis, an intermediate meeting will be needed to evaluate the project up to that point.
- ***M4*: Final meeting.** Prior to the final oral presentation, a last meeting will take place to discuss the last details and possible weaknesses of the report, and check the accomplishment of the objectives.

According to the Agile methodology (Section A), this project is organized by sprints, that is why a large number of tasks will take around a week, since it is the duration of those sprints. There is then a clear relation between the tasks and the sprints.

Considering all the stated hours, they represent a total of 510 hours of work. Nevertheless, this project is prone to cause problems and delays throughout its development, so there is a high probability that this number may increase.

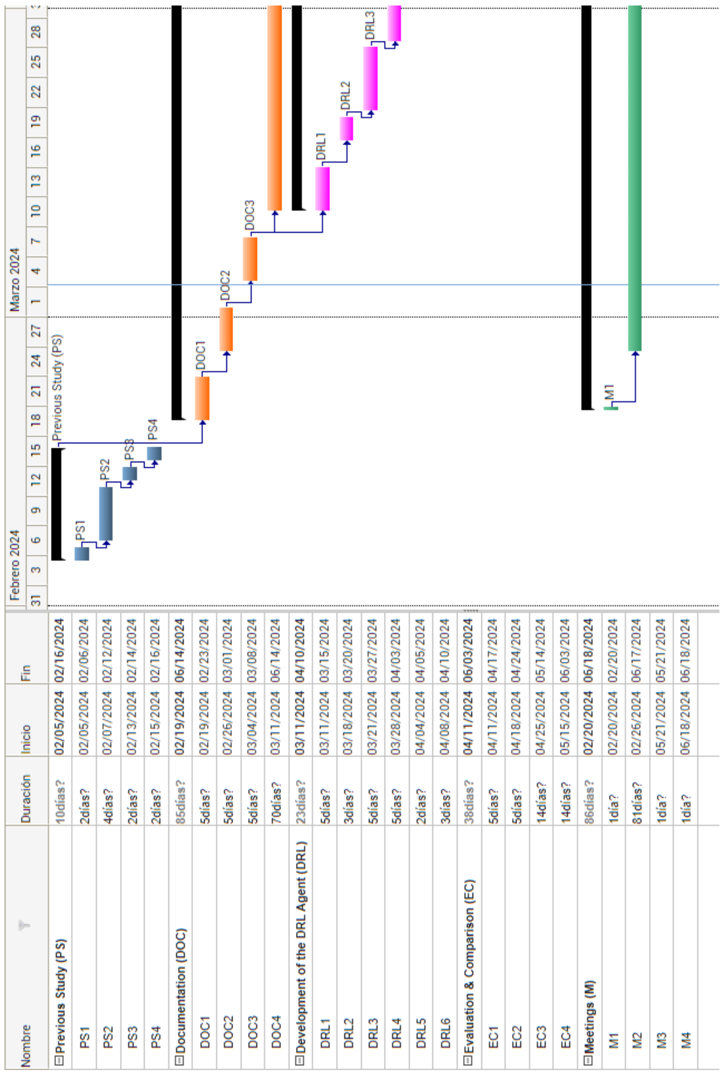
B.2 Overview of tasks

Figures B.1 and B.2 depicts a Gantt diagram showing the planning of the tasks. The arrows represent the relations of type *Finish-to-Start* of the tasks.

The time extension of them is represented by rectangles of different colours:

- *Black.* Represent the groups of tasks: *PS, DOC, DRL, EC, M*
- *Blue.* Represent the tasks that belong to the group *PS*.
- *Orange.* Represent the tasks of the group *DOC*, all of which start after the completion of the tasks of type *PS*.

- *Pink.* Represent the tasks of type *DRL*, starting alongside *DOC4*.
- *Yellow.* Represent the tasks of the group *EC*, which starts after the completion of the group *DRL*.
- *Green.* Represent the tasks of the group *M*, which represents all the meetings throughout the development of the project.



Nombre	Duración	Inicio	Fin
⊟ **Previous Study (PS)**	10días?	**02/05/2024**	**02/16/2024**
PS1	2días?	02/05/2024	02/06/2024
PS2	4días?	02/07/2024	02/12/2024
PS3	2días?	02/13/2024	02/14/2024
PS4	2días?	02/15/2024	02/16/2024
⊟ **Documentation (DOC)**	85días?	**02/19/2024**	**06/14/2024**
DOC1	5días?	02/19/2024	02/23/2024
DOC2	5días?	02/26/2024	03/01/2024
DOC3	5días?	03/04/2024	03/08/2024
DOC4	70días?	03/11/2024	06/14/2024
⊟ **Development of the DRL Agent (DRL)**	23días?	**03/11/2024**	**04/10/2024**
DRL1	5días?	03/11/2024	03/15/2024
DRL2	3días?	03/18/2024	03/20/2024
DRL3	5días?	03/21/2024	03/27/2024
DRL4	5días?	03/28/2024	04/03/2024
DRL5	2días?	04/04/2024	04/05/2024
DRL6	3días?	04/08/2024	04/10/2024
⊟ **Evaluation & Comparison (EC)**	38días?	**04/11/2024**	**06/03/2024**
EC1	5días?	04/11/2024	04/17/2024
EC2	5días?	04/18/2024	04/24/2024
EC3	14días?	04/25/2024	05/14/2024
EC4	14días?	05/15/2024	06/03/2024
⊟ **Meetings (M)**	86días?	**02/20/2024**	**06/18/2024**
M1	1día?	02/20/2024	02/20/2024
M2	81días?	02/26/2024	06/17/2024
M3	1día?	05/21/2024	05/21/2024
M4	1día?	06/18/2024	06/18/2024

Figure B.1: Gantt diagram. Part 1.

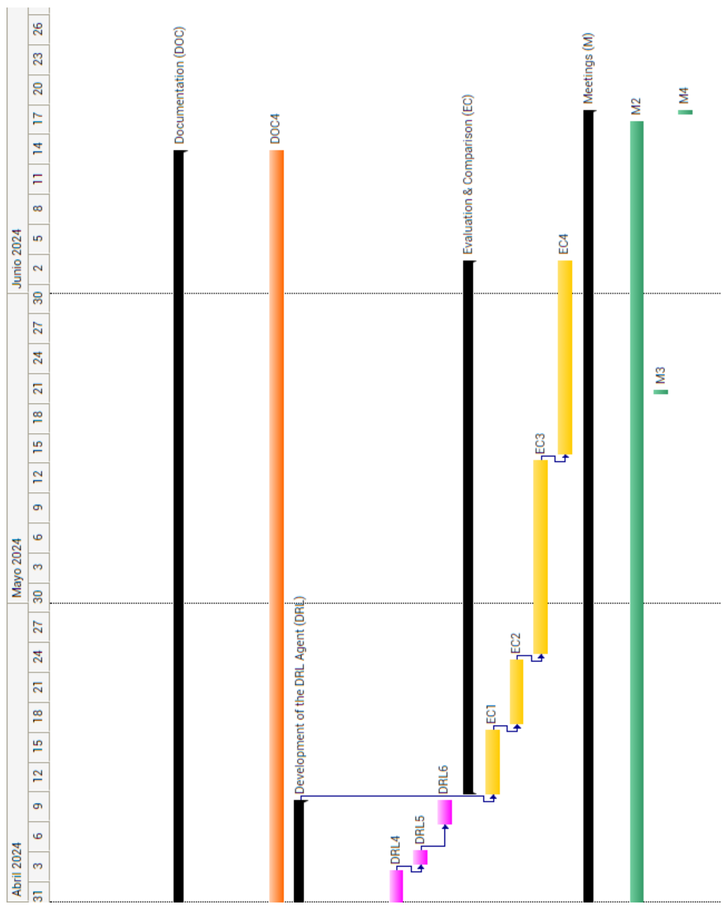


Figure B.2: Gantt diagram. Part 2.

Table B.1 summarizes the tasks, their respecting working hours, and the roles involved in each task, described in Section C.1. All tasks require just a computer, except *EC* tasks, all of which require in addition an external server (Section C.2).

ID	Task Name	Working Hours	Roles
PS1	Quantum context	8	ML, PS
PS2	Deep Reinforcement Learning	16	ML, PS
PS3	Graph Neural Networks	8	ML, PS
PS4	Merging GNN with DRL	8	ML, PS
DOC1	Contextualization and definition of the Scope	10	ML, PS
DOC2	Temporal planning	10	ML, PS
DOC3	Economic Management & Sustainability	10	ML, PS
DOC4	Documentation of the development of the project	140	ML
DRL1	Design the state space	20	ML
DRL2	Design the action space	12	ML
DRL3	Integration with GNN	20	ML
DRL4	Develop reward structure formulation of the DRL agent	20	ML
DRL5	Design termination conditions of the DRL agent	8	ML
DRL6	Implement the DRL environment dynamics	12	ML
EC1	Baseline evaluation using FGP-rOEE algorithm	20	ML
EC2	Set up synthetic and real circuits	20	ML
EC3	DRL agent training	75	ML
EC4	Evaluate and Compare	75	ML
M1	Initial meeting	1	ML, PS
M2	Regular monitoring meeting	16	ML, PS
M3	Intermediate evaluation meeting	1	ML, PS
M4	Final meeting	1	ML, PS

Table B.1: Hours and involved roles per task. *ML: Machine Learning Engineer, PS: Project Supervisor*

B.3 Temporal planning modifications

Although this work was expected to end by the 18th of June, some factors have forced to extend the end date. Specially, severe problems have been found with the efficiency of the presented algorithm. Therefore, more time than expected has been dedicated to solve this issue.

The deadline has been postponed to October. The extra time has been devoted to further improving the efficiency of the algorithm until reaching a feasible training time, since it has been impossible to train the model correctly with extremely long training times. Once the model achieved a sufficient level of efficiency in June, the training and evaluating phase began.

Appendix C

Resources

C.1 Human resources

All the parts and tasks of this project will be done by me, Arnau Esteban, taking the role of a junior machine learning engineer. Nevertheless, I will also have the support of my thesis supervisor, Sergi Abadal Cavallé, and my co-supervisor Pau Escofet. Both of them will be available in every weekly meeting to solve my issues, give me feedback and guide the development of my project. Therefore, they will serve as project supervisors. Besides the weekly meetings, they will also be available for on-demand assistance.

C.2 Material resources

This project is mainly based on research, so the principle resource will be numerous articles, papers and previous thesis.

Referring to hardware, I will use my personal computer for the research and for writing the documentation with OverLeaf. Hence, I will also need a space to work with Internet connection.

The training of the DRL agent will need a better-performing hardware than my personal computer, so an external server will be needed for this purpose.

Appendix D

Risk management

The final date estimation is the 3rd of June, which is three weeks before the final lecture date. This margin of time is necessary to resolve any unforeseen circumstances.

D.1 Long training time

As stated in Section 1.8, one of the possible setbacks of this research is the unexpectedly long training time. For instance, the training time of Yu Xia's thesis [9] was around 12-24 hours, but in this thesis, due to its higher complexity, the training time could be even higher.

- **Solution.** A possible solution for this issue could be upgrading the hardware used for this purpose. *Google Collab* is a free cloud computing platform that could be used to scale up the hardware, or *Amazon Web Services (AWS)* is another paid alternative.
- This change of hardware plus training again the model could potentially increase the working time by 20 hours.

Upgrading the hardware is not a mandatory alternative, it is possible to continue using the same hardware, but the training time would be higher.

D.2 DRL model not learning

Another risk of this project is the fact that the model does not sufficiently learn, leading to undesired results that do not fulfill the objectives (Section 1.4).

- **Solution.** Changing the configuration of the model by re-adjusting the hyperparameters, or changing the loss function and other characteristics that have influence on the model's results.
- These changes may increase the working time by approximately 30 hours.

Appendix E

Budget

E.1 Human costs

In this project, two main roles can contribute to its success: a machine learning engineer and a project supervisor. Additionally, other roles could be considered, like the project co-supervisor or other casual advisors for specific cases.

According to [29] and [30], the average base salaries in Spain of these two roles are the following (Table E.1).

Job	**Salary**
Junior Machine Learning Engineer	47669 €
Project Supervisor	46600 €

Table E.1: Average base salaries of a junior machine learning engineer and a project supervisor per year in Spain, according to [29] and [30]

This project is divided into hours, so it is necessary to extract the cost per hour of these roles. There are 2080 full-time working hours a year, so the salaries per hour are the following.

$$\frac{46600\,€}{2080\,\text{hours}} \approx 22.40\,€/\text{hour (Project Supervisor)}$$

$$\frac{47669\,€}{2080\,\text{hours}} \approx 22.92\,€/\text{hour (Junior Machine Learning Engineer)}$$

Considering the task schedule in Table B.1, Table E.2 describes the cost of human resources per task, and the total cost. All calculations have been done taking into account the salaries of each involved role. Additionally, the column

Cost w/ SS adjusts the cost by considering the Social Security tax, which implies multiplying the cost by 1.3.

The overall cost of salaries is then 17817.44 € (Social Security included).

ID	Hours	Roles	Cost	Cost w/ SS
PS	**40**	**-**	**1812.80 €**	**2356.64 €**
PS1	8	ML, PS	362.56 €	471.33 €
PS2	16	ML, PS	725.12 €	942.66 €
PS3	8	ML, PS	362.56 €	471.33 €
PS4	8	ML, PS	362.56 €	471.33 €
DOC	**170**	**-**	**4568.40 €**	**5938.92 €**
DOC1	10	ML, PS	453.20 €	589.16 €
DOC2	10	ML, PS	453.20 €	589.16 €
DOC3	10	ML, PS	453.20 €	589.16 €
DOC4	140	ML	3208.80 €	4170.44 €
DRL	**92**	**-**	**2108.64 €**	**2741.23 €**
DRL1	20	ML	458.40 €	596.52 €
DRL2	12	ML	275.04€	357.55 €
DRL3	20	ML	458.40 €	596.52 €
DRL4	20	ML	458.40 €	596.52 €
DRL5	8	ML	183.36 €	238.35 €
DRL6	12	ML	275.04 €	357.55 €
EC	**190**	**-**	**4354.80 €**	**5661.24 €**
EC1	20	ML	458.40 €	596.52 €
EC2	20	ML	458.40 €	596.52 €
EC3	75	ML	1719 €	2234.70 €
EC4	75	ML	1719 €	2234.70 €
M	**19**	**-**	**861.08 €**	**1119.40 €**
M1	1	ML, PS	45.32 €	58.92 €
M2	16	ML, PS	725.12 €	942.66 €
M3	1	ML, PS	45.32 €	58.92 €
M4	1	ML, PS	45.32 €	58.92 €
Total	**511**	**-**	**13795.72 €**	**17817.44 €**

Table E.2: Summary of human costs. *ML: Machine Learning Engineer, PS: Project Supervisor, Cost w/ SS: Cost with Social Security.*

E.2 Generic costs

In this project, the *Visual Studio Code* software will be used, which is at zero cost, so it will not be considered in the budget. Furthermore, if no further problems appear, external hardware will be used with *Google Collab*, which is also not considered in the budget since it is free.

Hardware

As stated in Section C.2, in this project, a personal computer will be used, which is a M1 MacBook Pro with 16GB of RAM, whose price when it was launched was 1699 €, and has an estimated life expectancy of seven years [31]. It is expected to be used daily for all the tasks during the 4 months of the project.

Its depreciation is calculated as follows:

$$\frac{1699\,€}{7\,\text{years}} \times \frac{1\,\text{year}}{12\,\text{months}} \times 4\,\text{months} = 80.90\,€$$

Space and consumption costs

The project will be carried out daily at home, a shared space, in Castelldefels. To estimate the cost of it, the price of an equivalent *co-working* space in Castelldefels will be considered, which is 300 euros per month [32], including available access every day, electricity, a personal desk, and internet. Therefore, taking into account that the project will take 4 months, the total price is 1200 euros.

Total

Table E.3 summarizes the generic costs explained above.

Computer depreciation	80.90 €
Space and consumption	1200 €
Total	1280.90 €

Table E.3: Summary of generic costs

E.3 Contingency

The aforementioned budgets (Sections E.1 and E.2) are simply estimations, so there is a risk of cost overrun. In order to have a security margin, it is crucial to have a contingency fund. In this case, an extra charge of 10% is established, which represents 1917.74 euros (Table E.4).

Human costs	17817.44 €
Generic costs	1280.90 €
Total	19098.34 €
Contingency (10%)	1909.83 €
Total w/ Contingency	21008.17 €

Table E.4: Summary of Human and Generic costs with a Contingency of 10%.

E.4 Incidentals

In the event of any incident caused by the risks mentioned in Section 1.8, it would have an effect on the budget. The potential extra charges are described in this section.

E.4.1 Long training time

If the training time is excessively long, it will be necessary to use paid cloud computing platforms, such as AWS. Their price may vary considerably depending on the needed hardware and the hours of usage. Thus, in order to have a wider margin, the price in this case will be estimated upwards. This event has a probability of 30% to happen and would require additional 20 hours of work to the machine learning engineer, and would also imply paying the cloud computing service.

In this case, an estimation of 350 euros for the cloud computing platform has been done. Additionally, 20 hours of work for the machine learning engineer, based on the salary stated in Section E.1, represent a cost of $22.92\,€ \times 20\,\text{hours} = 458.40\,€$.

In total, in this unforeseen circumstance, the cost overrun would be:

$$350\,€ + 458.40\,€ = 808.40\,€$$

E.4.2 DRL model not learning

If the DRL model does not learn and the results are not aligned with the objectives of the project (Section 1.4), 30 additional working hours will be needed for the machine learning engineer to figure out the re-adjustment of the model. This has a likelihood of 20% to happen.

In this case, the cost overrun would be:

$$22.92\,€ \times 30\,\text{hours} = 687.60\,€$$

E.4.3 Incidentals table

Table E.5 describes the extra costs of incidents. The price of an extra cost is multiplied by its probability to happen.

Incident	**Estimated cost**	**Probability**	**Cost**
Long training time	808.40 €	30%	242.52 €
DRL model not learning	687.60 €	20%	137.52 €
Total	-	-	**380.04 €**

Table E.5: Summary of incidental extra charges

E.5 Overall budget

Once all the budgets have been described, Table E.6 depicts the final budget of the project. The total cost is 21475.21 €.

Type	Cost
Human costs	17817.44 €
Computer depreciation	80.90 €
Space and consumption costs	1200 €
Contingency	1909.83 €
Incidentals	380.04 €
Total	21388.21 €

Table E.6: Summary of incidental extra charges

E.6 Management control

Throughout the development of this project, it is essential to keep track of the budget management. For this purpose, some metrics will be used to calculate the deviation from the estimated hours and costs.

Every time a task is completed, the deviation in terms of hours and costs will be calculated, and therefore, the budget will be re-adjusted if necessary. If there have been extra charges, the source of the problem will be analyzed and the necessary actions will be taken in order to counteract future incidents.

The following metrics will be used to calculate the deviations:

1. **Personal cost deviation per task**
   *(estimated_cost - real_cost) * real_hours*

2. **Cost deviation due to hours of work**
   *(estimated_hours - real_hours) * real_cost*

3. **Total cost deviation**
   total_estimated_cost - total_real_cost

4. **Total hours deviation**
   total_estimated_hours - total_real_hours

Metrics 3 and 4 can also be used in segmented parts of the budget, such as only for personal resources, for generic costs, among others.

Appendix F

Sustainability

F.1 Reflection

In my personal case, since I was young, I have always been surrounded by an environment where sustainability has been a recurrent topic, mostly at school. Nevertheless, I have realized that my knowledge about it is not enough, since I am not entirely aware of this issue when it is related to the tech industry, where, in fact, regulations are increasing, and most of us are not conscious of the environmental effect behind our screens.

Personally speaking, most people are nowadays concerned about sustainability, but we are only aware of direct actions that affect the environment. However, we should raise awareness of the actions that indirectly worsen the situation too. For instance, using or training AI models implies the use of enormous supercomputers, which require a lot of energy that is possibly coming from a non-renewable source.

As technology continuously advances at an unprecedented pace, and therefore the hardware demand is increasing, this represents a potential risk of environmental impact. Hence, I feel highly responsible for carrying out a sustainable initiative in this new era, considering all the direct and indirect effects of my projects on the carbon footprint, and trying to reduce it at the lowest possible level.

I am currently becoming more and more conscious of the importance of measuring the environmental impact of any emerging enterprise or project, and keeping up with the newest regulations to guarantee that new projects respect the three important dimensions: economy, society and environment. That is why, not only is it crucial to have a deeper knowledge of this issue in terms of awareness, but also in terms of indirect carbon footprint, current regulations and measuring tools, in order to stop being an observer to become an actor.

F.2 Economic dimension

The estimated cost for the completion of this project is coherent with the problem it solves. Quantum computing is a new area where large amounts of money are being invested, and there are still countless challenges to face. This project solves one of them that is, to date, intractable if it is scaled up to real scenarios. Therefore, the potential of this project is promising, and the added value of the solution can be very profitable for other research groups and *big techs* to continue their development of an efficient quantum computer.

As aforementioned, the problem that this thesis solves (Section 1.3) is intractable, so the hardware resources utilized for this purpose need to be advanced and require substantial computation power, leading to large economic expenses. The DRL model that is created in this thesis is scalable and more efficient, hugely reducing the hardware and power needed. Consequently, the cost of executing this model greatly decreases compared to the state-of-the-art solution.

F.3 Environmental dimension

The use of hardware resources to train and execute the resulting model have a direct and indirect impact on the environment. Going into detail, the utilized hardware requires a large quantity of electricity, it needs to be refrigerated, needs maintenance, and its manufacturing process also causes an environmental footprint.

This project is composed of two phases with different environmental impacts.

Training. The training phase of the model is clearly the step where the hardware consumption is the highest. During this phase, numerous power-demanding experiments will be conducted, leading to a huge electricity consumption. Therefore, this part of the project is the one where the environmental impact is the highest. Nevertheless, the strong point is that this step will only be effectuated once to adjust the model.

Execution. Once the model has been trained, it can be used whenever desired. The execution of the model is much more efficient than the previous phase, and more efficient than state-of-the-art solutions. This is a great advantage since the environmental footprint is, hence, minor.

During the training phase, it is difficult to reduce the environmental impact, since it is the most important step of the project and, due to the problem's nature, it is hard to reduce the consumption. However, efforts will be made to use the most efficient training techniques.

On the other hand, this early environmental impact comes with good advantages since, once the model will be trained, the required hardware to execute the model in the future will be even minor than the state-of-the-art solutions, leading to the use of even more efficient and less power-consuming hardware. The solution reduces energy consumption by approximately 30 times, thanks to an AI agent trained for quantum computer compilation. Additionally, quantum compilation itself exponentially enhances the efficiency of certain programs. This represents a strong point for sustainability, at the expense of the initial costs.

Overall, once the model is trained, this project offers a great advantage over other solutions since it is more efficient, and therefore it has a smaller environmental footprint than state-of-the-art solutions, which require a large amount of electricity due to the intractability of the problem.

F.4 Social dimension

Personally speaking, throughout my Bachelor's degree program, I have realized that the Artificial Intelligence realm and its real-world applications are my main interests. Therefore, this project will help me gain deeper knowledge and experience of this field, as well as a more profound comprehension of the environmental impact.

As aforementioned in Section 1.7, the direct stakeholders in this project are specialised research groups, which can use this thesis' solution to continue their development of efficient quantum computers. However, indirectly, this project can boost the production and adoption of quantum computers, leading to new applications that have the potential to revolutionise a large number of industries, highly benefiting people's lives.

Bibliography

[1] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Journal on Computing*, vol. 26, no. 5, pp. 1484–1509, 1997. [Online]. Available: https://doi.org/10.1137/S0097539795293172

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

[3] "IBM Launches $100 Million Partnership with Global Universities to Develop Novel Technologies Towards a 100,000-Qubit Quantum-Centric Supercomputer." [Online]. Available: https://newsroom.ibm.com/2023-05-21-IBM-Launches-100-Million-Partnership-with-Global-Universities-to-Develop-Novel-Technologies-Towards-a-100,000-Qubit-Quantum-Centric-Supercomputer

[4] "IBM Quantum Computing Blog | IBM Quantum System Two: the era of quantum utility is here." [Online]. Available: https://www.ibm.com/quantum/blog/quantum-roadmap-2033

[5] C. Gidney and M. Ekerå, "How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits," *Quantum*, vol. 5, p. 433, Apr. 2021, publisher: Verein zur Förderung des Open Access Publizierens in den Quantenwissenschaften. [Online]. Available: https://quantum-journal.org/papers/q-2021-04-15-433/

[6] J. M. Baker, C. Duckering, A. Hoover, and F. T. Chong, "Time-sliced quantum circuit partitioning for modular architectures," in *Proceedings of the 17th ACM International Conference on Computing Frontiers*, ser. CF '20. New York, NY, USA: Association for Computing Machinery, May 2020, pp. 98–107. [Online]. Available: https://dl.acm.org/doi/10.1145/3387902.3392617

[7] S. Rodrigo, S. Abadal, E. Alarcon, M. Bandic, H. Someren, and C. G. Almudever, "On double full-stack communication-enabled architectures for multicore quantum computers," *IEEE Micro*, vol. 41, no. 05, pp. 48–56, sep 2021.

[8] A. Pastor Lacueva, "Towards Scalable Circuit Partitioning for Multi-Core Quantum Architectures with Deep Reinforcement Learning," Bachelor thesis, Universitat Politècnica de Catalunya, Jun. 2023, accepted: 2023-10-04T09:29:29Z. [Online]. Available: https://upcommons.upc.edu/handle/2117/394559

[9] R. Yu Xia, "GNN for time-sliced quantum circuit partitioning," Master's thesis, Universitat Politècnica de Catalunya, Jan. 2022, accepted: 2022-03-10T14:11:42Z. [Online]. Available: https://upcommons.upc.edu/handle/2117/363859

[10] S. Boixo, S. V. Isakov, V. N. Smelyanskiy, R. Babbush, N. Ding, Z. Jiang, M. J. Bremner, J. M. Martinis, and H. Neven, "Characterizing quantum supremacy in near-term devices," *Nature Physics*, vol. 14, no. 6, pp. 595–600, Jun. 2018. [Online]. Available: https://doi.org/10.1038/s41567-018-0124-x

[11] D. Ferrari, A. S. Cacciapuoti, M. Amoretti, and M. Caleffi, "Compiler design for distributed quantum computing," *IEEE Transactions on Quantum Engineering*, vol. PP, pp. 1–1, 01 2021.

[12] Z. Krunic, F. Flother, G. Seegan, N. Earnest-Noble, and S. Omar, "Quantum kernels for real-world predictions based on electronic health records," *IEEE Transactions on Quantum Engineering*, vol. 3, pp. 1–1, 01 2022.

[13] P. Escofet, A. Ovide, M. Bandic, L. Prielinger, H. van Someren, S. Feld, E. Alarcón, S. Abadal, and C. G. Almudéver, "Revisiting the mapping of quantum circuits: Entering the multi-core era," *ACM Transactions on Quantum Computing*, Mar. 2024, just Accepted. [Online]. Available: https://doi.org/10.1145/3655029

[14] P. Garnier, J. Viquerat, J. Rabault, A. Larcher, A. Kuhnle, and E. Hachem, "A review on deep reinforcement learning for fluid mechanics," 08 2019.

[15] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3, pp. 279–292, May 1992. [Online]. Available: https://doi.org/10.1007/BF00992698

[16] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015. [Online]. Available: https://doi.org/10.1038/nature14236

[17] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57–81, Jan. 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666651021000012

[18] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. PMLR, 06–11 Aug 2017, pp. 1263–1272. [Online]. Available: https://proceedings.mlr.press/v70/gilmer17a.html

[19] M. Bandic, L. Prielinger, J. Nüßlein, A. Ovide, S. Rodrigo, S. Abadal, H. van Someren, G. Vardoyan, E. Alarcon, C. G. Almudever, and S. Feld, "Mapping quantum circuits to modular architectures with QUBO," in *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, Sep. 2023, pp. 790–801, arXiv:2305.06687 [quant-ph]. [Online]. Available: http://arxiv.org/abs/2305.06687

[20] P. Escofet, A. Ovide, C. G. Almudever, E. Alarcón, and S. Abadal, "Hungarian Qubit Assignment for Optimized Mapping of Quantum Circuits on Multi-Core Architectures," *IEEE Computer Architecture Letters*, vol. 22, no. 2, pp. 161–164, Jul. 2023, arXiv:2309.12182 [quant-ph]. [Online]. Available: http://arxiv.org/abs/2309.12182

[21] T. Park and C. Y. Lee, "Algorithms for partitioning a graph," *Computers Industrial Engineering*, vol. 28, no. 4, pp. 899–909, 1995. [Online]. Available: https://www.sciencedirect.com/science/article/pii/036083529500003J

[22] A. Ovide, S. Rodrigo, M. Bandic, H. Van Someren, S. Feld, S. Abadal, E. Alarcon, and C. G. Almudever, "Mapping quantum algorithms to multi-core quantum computing architectures," in *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2023, pp. 1–5.

[23] G. Pascoal, J. Fernandes, and R. Abreu, "Deep reinforcement learning strategies for noise-adaptive qubit routing," in *2024 IEEE International Conference on Quantum Software (QSW)*. Los Alamitos, CA, USA: IEEE Computer Society, jul 2024, pp. 146–156. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/QSW62656.2024.00030

[24] D. Kremer, V. Villar, H. Paik, I. Duran, I. Faro, and J. Cruz-Benito, "Practical and efficient quantum circuit synthesis and transpiling with reinforcement learning," May 2024. [Online]. Available: https://arxiv.org/abs/2405.13196v1

[25] O. Ramos Núñez, "Synthetic quantum algorithm generator for the benchmarking of quantum computers," Master's thesis, Universitat Politècnica de Catalunya, 2024.

[26] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray,

C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[27] C. Laidlaw, B. Zhu, S. Russell, and A. Dragan, "The effective horizon explains deep RL performance in stochastic environments," Dec. 2023.

[28] C. Laidlaw, S. Russell, and A. Dragan, "Bridging RL theory and practice with the effective horizon," Apr. 2023.

[29] E. E. R. Institute, "Machine Learning Engineer Salary Spain - SalaryExpert." [Online]. Available: https://www.salaryexpert.com/salary/job/machine-learning-engineer/spain

[30] ——, "Supervisor Project Salary Madrid, Spain - SalaryExpert." [Online]. Available: https://www.salaryexpert.com/salary/job/supervisor-project/spain/madrid

[31] "How Long Does MacBook Pro Last? (Lifespan Explained)," Feb. 2022, section: Tips. [Online]. Available: https://www.macbookproslow.com/how-long-macbook-pro-last/

[32] "Coworking en Castelldefels | 860 Coworking." [Online]. Available: https://860coworking.com/